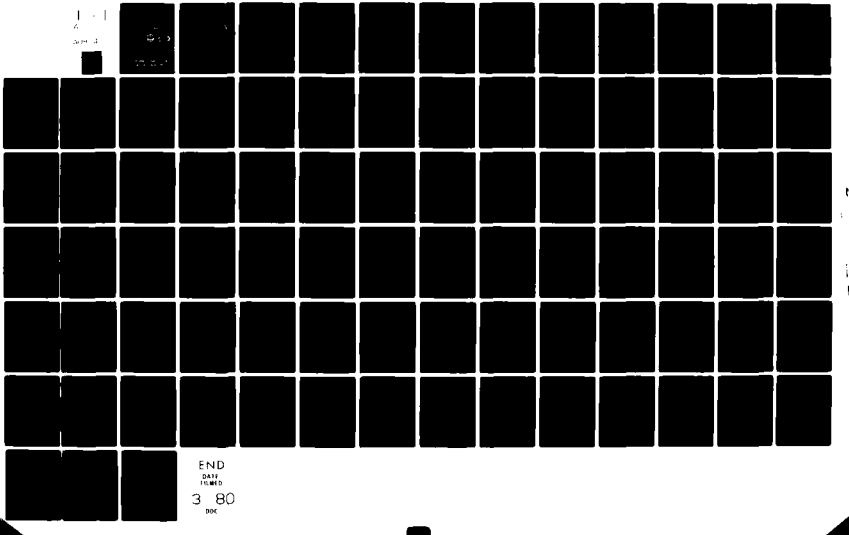
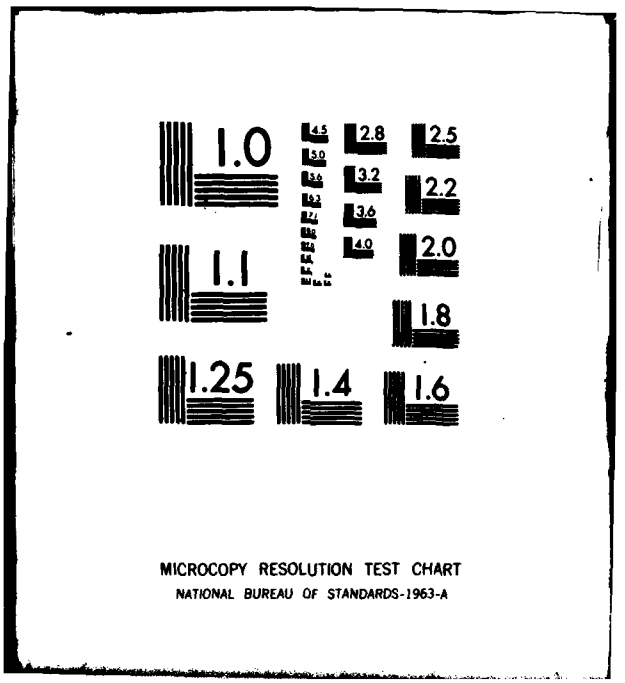
AD-A081 247 GEORGIA INST OF TECH ATLANTA SCHOOL OF INFORMATION A--ETC F/G 12/1
COMBINATORIAL GRAPH EMBEDDINGS.(U)
JAN 80 R A DEMILLO, S C EISENSTAT, R J LIPTON DAAG29-76-G-0338
UNCLASSIFIED GIT-ICS-79/12 ARO-14690.10-EL NL



END
DATE FILMED
3 80
DDC


1.0
1.1
1.25
1.4
1.6
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.8
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

LEVEL II

See 1475 in back

School of
Information and Computer Science

GEORGIA INSTITUTE
OF TECHNOLOGY

12

GIT-ICS-79/12

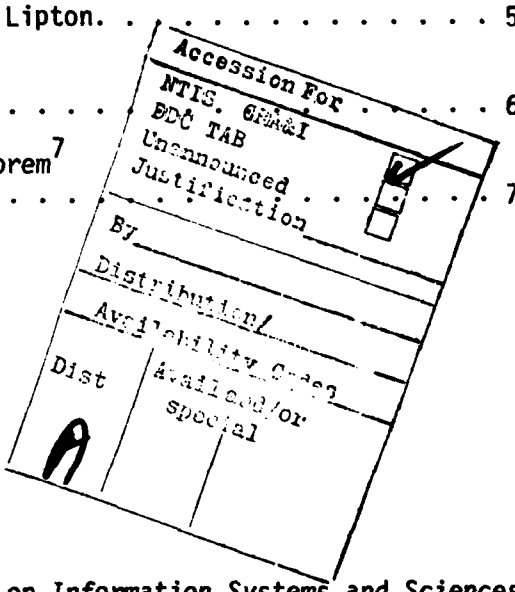
COMBINATORIAL GRAPH EMBEDDING

R.A. DeMillo
S.C. Eisenstat
R.J. Lipton

January, 1980

Final Report: ARO Grant No. DAAG29-76G-0338

Contents



[1] JACM, 23(4)(October, 1976) pp. 720-732

[2] JACM, 21(1)(Jan,1980)

[3] Submitted for Publication

[4] CACM, 21(3)(March,1978) pp. 228-231

[5] Proceedings 1978 Johns Hopkins Conference on Information Systems and Sciences, Baltimore, Md., March 1978, pp. 421-428

[6] SIAM J. Appl. Math, 36(2)(April,1979) pp. 177-189

[7] 1977 FOCS Conference, Providence, R.I., November 1977, pp. 162-170

INTRODUCTION

Let G, G' be directed graphs. A combinatorial embedding of G into G' is an identification of each $x \in V(G)$ with a set of vertices $S \subseteq V(G')$ such that each S is bounded in size by a constant independent of $|V(G)|$ and each arc in G is carried into a directed path of length bounded by a constant independent of $|V(G)|$. This concept (first defined in [A]) has formed the basis for a number of theoretical studies supported by ARO Contract No. DAAG29-76-G-0338, and the papers collected here are representative of - with one major exception - the state-of-the-art with regard to graph embeddings.

First, a word regarding the subject matter of these papers. By modelling the control structures of programs as classes of directed graphs, asymptotic properties of control structure transformations can be obtained. This is the principle aim of [A,B,C]. Knuth [1] surveys a number of results concerning control structure transformations and places the graph embedding results in context. Directed graphs also model data storage structures (vertices model nodes or records, arcs model logical adjacencies). The notion of graph embedding can be used to compare the relative storage efficiencies of classes of data structures [D,E,F]. Several researchers have attempted to generalize these results to more encompassing notions of data storage and representation (see e.g., [2,7]) and more sophisticated types of analysis [3]. The purely combinatorial notions involved in data structure embeddings also make contact with a variety of other theoretical and numerical problems [4]. In fact, one of the principle devices used in the results of [A-F] is the notion of "cutting" graphs along boundaries of connected regions. A boundary which cuts a graph is called a separator and in [G,H] a characterization of separator graphs is derived and used to obtain results in areas from Turing Machine complexity to optimization theory.

Graph Embeddings, boundary arguments and graph theoretical models of computation all appear to be related in sometimes surprising ways [5,6,7,8]. Missing from the collection is a coherent account of these connections. It will have to suffice that the connections run deeper than the surface. We anticipate reporting on this aspect of graph embedding elsewhere.

References

1. D.E. Knuth
"Structured Programming with goto Statements" in R.Yeh, editor, Current Trends in Programming Methodology, Volume I, Software Specification and Design, Prentice-Hall, 1977, pp . 140-194.

2. A.L. Rosenberg
"Data Encodings and Their Costs", Acta Informatica, Vol. 9, 1978, pp. 273-292.

3. A.L. Chow
"Preserving Average Proximity in Arrays with Duplication", M.S. Thesis, University of Illinois, Urbana, Report No. R-812, Coordinated Science Laboratory.

4. M.R. Garey, R.L. Graham, D.S. Johnson, and D.E. Knuth
"Complexity Results for Bandwidth Minimization", J. Combinatorial Theory, (to appear).

5. L.G. Valiant
"Negation can be Exponentially Powerful", Proceedings 11th ACM Symposium on Theory of Computing, 1979, pp. 189-196.

6. G.S. Tseitin
"On the Complexity of Derivations in the Propositional Calculus" in A.O. Slisenko, editor, Structures in Constructive Mathematics and Mathematical Logic, 1968, pp. 115-125.

7. A. George
"Nested Dissection of a Regular Finite Element Mesh", SIAM J. Numerical Anal., Vol. 10, No. 2, April 1973, pp. 345-363.

8. S. Cook
"Observation of a Storage-Time Tradeoff", Proc. 5th ACM Symposium on Theory of Computing, 1973, pp. 29-33.

Space and Time Hierarchies for Classes of Control Structures and Data Structures

R. J. LIPTON AND S. C. EISENSTAT

Yale University, New Haven, Connecticut

AND

R. A. DeMILLO

University of Wisconsin, Milwaukee, Wisconsin

ABSTRACT. Control structures and data structures are modeled by directed graphs. In the control case nodes represent executable statements and arcs represent possible flow of control; in the data case nodes represent memory locations and arcs represent logical adjacencies in the data structure. Classes of graphs are compared by a relation $\leq_{S,T}$ where $G \leq_{S,T} H$ if G can be embedded in H with at most a T-fold increase in distance between embedded nodes by making at most S "copies" of any node in G. For both control structures and data structures, S and T are interpreted as space and time constants, respectively. Results are presented that establish hierarchies with respect to $\leq_{S,T}$ for (1) data structures, (2) sequential program schemata normal forms, and (3) sequential control structures.

KEY WORDS AND PHRASES: ancestor tree, bounded simulation, complexity, control structure, data structure, directed graph, **do forever** program, embedding, **go to** program, label **exit** program, normal form programs, structured programming, **while** programs

CR CATEGORIES: 4.22, 4.34, 5.24, 5.25, 5.32

1. *Introduction*

The running time or computational complexity of a sequential process is usually estimated by summing weights attached to the basic operations from which the process is derived. In practice, however, the complexity of a program is often limited by how efficiently it can access its data structures and control program flow. Furthermore, it has been extensively argued [4] that certain limitations on the process sequencing mechanisms available to the programmer result in more "efficient" representations for the underlying processes. In this paper we examine these issues in an attempt to assess the "power" of various data and control structures.

A key observation about sequential processes is that they usually do not reference

Some of these results were presented at the 7th ACM Symposium on the Theory of Computing, Albuquerque, New Mexico, May 1975.

The research was supported in part by the US Army Research Office under Grants DAHC04-74-G-0179 and DAHC04-75-G-0037 and by the Office of Naval Research under Grant N00014-67-0097-0016. Some of this work was carried out while DeMillo and Lipton were visitors at the Communications/ADP Laboratory of the US Army Electronics Command, Fort Monmouth, New Jersey.

Authors' addresses: R.J. Lipton and S.C. Eisenstat, Department of Computer Science, Yale University, 10 Hillhouse Ave., New Haven, CT 06520; R.A. DeMillo, School of Information and Computer Science, Georgia Institute of Technology, Atlanta, GA 30332.

their data randomly. For instance, algorithms that organize their data structures as arrays often access the array elements in a "local" manner (e.g. the conventional matrix multiplication algorithm accesses its arrays by rows and by columns). Thus in a paging environment how one stores an array is especially important (cf. Moler [13], Rosenberg [16]), and it is natural to investigate how arrays can be stored so that elements "near" one another in the array are stored near one another in memory. Data structures are compared by the relation $\leq_{1,T}$: For data structures G and G^*, $G \leq_{1,T} G^*$ if G can be embedded in G^* so that there is at most a T-fold increase in distance between embedded objects.

It is somewhat unexpected that an analogous study for control structures uses the same basic insights. It is well known that process sequencing disciplines found in programming practice (e.g. **go to, while**) can *simulate* each other and are thus equivalent in the sense of yielding *functionally* equivalent programs, but are inequivalent relative to the stronger requirement of structural isomorphism [1-3, 10, 11]. We argue that the fundamental issue is neither the construction of functionally equivalent programs nor the inability to preserve structure exactly, but rather the "naturalness" of the simulation. Control structures are compared by the relation $\leq_{S,T}$: For algorithms G and G^* with distinct process sequencing mechanisms, $G \leq_{S,T} G^*$ if G^* simulates G by making at most S copies of each operation in G and increasing the cost of sequential access of embedded operations by a factor of at most T.

Thus comparing the power of data structures and control structures involves analyzing the one-one and many-one aspects of embedding (or simulation) techniques whose efficiency is bounded by S and T. In a natural way, the relation $\leq_{S,T}$ represents an intertwining of space and time complexities.

In Section 2 basic combinatorial definitions are presented, and the combinatorial models used for representing data structures and control structures are introduced. In Section 3 the relation $\leq_{S,T}$ is defined by means of graph embeddings. This relation is viewed as an embedding in the data structure case and as a simulation relation in the control structure case. Section 4 contains the main result for data structure embeddings: For certain families of structures $\{G_i\}_{i\geq 0}$ and $\{G_i^*\}_{i\geq 0}$, if $G_i \leq_{1,T} G_i^*$, then $T \geq c \cdot \log n_i$ (see footnote 1) for some positive constant c whose choice is independent of n_i, the number of components of G_i.

The main theorem in Section 5 generalizes the result in Section 4 by allowing $S \geq 1$. In this case, if $G_i \leq_{S,T} G_i^*$ for certain natural choices of $\{G_i\}_{i\geq 0}$ and $\{G_i^*\}_{i\geq 0}$, then $T + \log S \geq c \cdot \log n_i$, where n_i is the number of components of G_i and c is a positive constant independent of n_i. A direct result of this theorem is that certain schema constructions, such as Engeler normal form [6], cannot be achieved "uniformly" with respect to the $\leq_{S,T}$ relation. More exactly, for any constants S and T there is a **go to** program G such that for no program H in Engeler normal form is $G \leq_{S,T} H$. Thus, the construction of Engeler normal forms—while always possible—does not preserve time and space in a bounded way. This result also demonstrates how our results will be asymptotic in their nature: For any **go to** program G there are S and T such that $G \leq_{S,T} H$ where H is the Engeler normal form; however, the values of S and T must grow with the size of the program G.

In Section 6 the relation $\leq_{S,T}$ is placed in the context of relations used in previous studies of control structure simulation. The main simulation results for control structures are then developed, giving rise to the hierarchy of control structures shown in Figure 1. An important result is that **go to** programs are strictly more powerful than label **exit** programs. Since the class of label **exit** programs includes many of the standard constructs that are allowed in "structured" programs, this result can be viewed as a precise sense in

[1] When we establish results of this form, we are asserting that there is a minimal rate of growth for T as a function of n. In the sequel we will consistently abuse our notation by writing $f(S, T) \geq g(n)$ instead of the less convenient $f(S(n), T(n)) \geq g(n)$. It will usually be clear from context when S, T are to be considered constants and when S and T are parameterized.

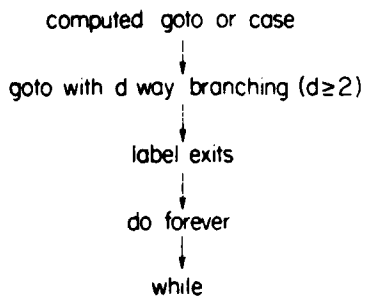


FIG. 1

which there is a *time-space speedup* between **go to** programs and "structured" programs: There are **go to** programs whose only "structured" counterparts explode in either time or space. This result seems to make precise the comments of Knuth [9] on the efficiency of **go to** and "structured" programs.

While the results in this paper are motivated by our interest in the power of data and control structures, they may have interest purely as combinatorial results.

2. *The Combinatorial Representations*

A *directed graph* G is an ordered pair (V, E) of *nodes* and *arcs*. If there is an arc from x to y and an arc from y to x, then we say there is an *edge* between x and y. Moreover, the arcs shown in Figure 2(a) are represented as in Figure 2(b). A *path* from x to y is defined by any sequence of arcs from $x = x_0$ to x_1 to $x_2, \ldots, x_{n-1}$ to $x_n = y$. We define a *metric* $d_G(x, y)$ on G as the number of arcs in a minimal length path from x and y.

A *binary tree*[2] is a finite set of nodes that either is a single node or consists of a root x and an edge between x and the root of each of two binary trees called the left and right *subtrees* of the root (cf. Knuth [8]). Note that nodes in a binary tree are connected by *edges* so that the metric is symmetric. If G is a binary tree, then a node x of G is a *leaf* of G if x has no sons.

We represent both control structures and data structures by directed graphs. In the control case, the nodes of a graph G represent executable statements and the arcs represent possible flow of control; in the data case, the nodes of the graph represent memory locations and the arcs represent logical adjacencies in the data structure. Thus in either case what is to be modeled is the "difficulty" of accessing nodes: The complexity of a control structure[3] is given by the cost of accessing and sequencing noncontrol instructions, while the complexity of a data structure is determined by the cost of accessing successive data elements. Each class of control structure or data structure can be studied in terms of restrictions on what graphs are allowed in that class.

2.1. DATA STRUCTURES. The two classes of data structure we deal with are *arrays* and *ancestor trees*.

Arrays. G_n denotes the data structure corresponding to an $n \times n$ array. If the nodes of G_n are indexed by (i, j) where $1 \le i \le n$ and $1 \le j \le n$, then there is an *edge* between (i, j) and $(i, j + 1)$, for $1 \le i \le n$, $1 \le j < n$, and between (i, j) and $(i + 1, j)$, for $1 \le i < n$, $1 \le j \le n$. Thus G_n is "rook connected." For instance, G_3 is illustrated in Figure 3.

Ancestor trees. Ancestor trees are binary trees with an additional feature: A node x of an ancestor tree may be connected by an *arc* to any of its ancestors. For example, the graph shown in Figure 4 is an ancestor tree because y is both an ancestor *and* a successor

[2] If x is a root with subtree H and y is a root of one of the subtrees of x, then y is a *son* of x; further, x is an *ancestor* of each node in H, while each node in H is a *successor* of x.

[3] Some care must be exercised in viewing control structures that are represented in this way; our representations do not always correspond to (temporal) flow of control and are not to be looked at as flowcharts. Rather, what is being modeled is the *potential* control connectivity of an underlying algorithm or process.

FIG. 2

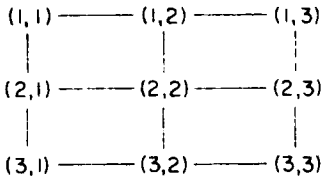


FIG. 3

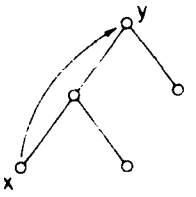


FIG. 4

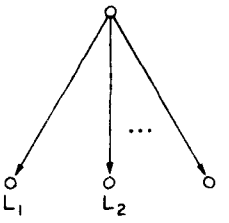


FIG. 5

of x. Notice, however, that unlike metrics on binary trees, the graph metric d_G is not necessarily symmetric on ancestor trees. Ancestor trees include linear lists, circular lists, and threaded lists [14] as special cases.

2.2. CONTROL STRUCTURES. We consider the following five classes of control structures: computed **go to**; **go to** with d-way branching; label **exit**; **do forever**; **while**. In addition, all of the available classes have access to a *sequential* flow of control and an *alternative* (e.g. **if-then-else**) flow of control. Since the constructions described below do not involve schema manipulation, the details of these features need not be made explicit. Indeed, there are a number of ways to represent these features in our model, and our later results are invariant under the differing representations. We now present the class of graphs that represent programs formed from each of the five control structures.

Computed **go to** *programs.* **go to**$_\omega$ programs are programs that allow arbitrary branching between statements. For instance, we allow for representations of the construct **go to** i $(L_1, \ldots, L_n)$, which branches to the ith label depending on the value of i. Thus this class of programs is represented by the entire class of directed graphs. The construct above is represented by the graph of Figure 5, a node with n arcs leading to nodes labeled by $L_1, \ldots, L_n$.

go to *programs with d-way branching.* **go to**$_d$ programs are programs in which the amount of branching that is possible in one step is bounded by the integer d. For example, the Fortran construct IF (E) L_1, L_2, L_3 falls in the class **go to**$_3$. Programs with d-way branching are represented by the class of directed graphs with a maximum out-degree d.[4]

Label **exit**, **do forever**, *and* **while** *programs.* Label **exit**, **do forever**, and **while** programs are defined as certain classes of ancestor trees. In order to define these classes, we need the following relations, which are defined for any ancestor tree:

$x \underset{p}{\rightarrow} y$ if y is the left son of x.

$x \underset{s}{\rightarrow} y$ if y is the right son of x.

$x \underset{a}{\rightarrow} y$ if y has an ancestor pointer from x.

[4] The out-degree of a node x is $\|\{y : d_G(x, y) = 1\}\|$.

We view $x \xrightarrow{p} y$ as meaning that statement x can "push" into a substructure with first statement y; $x \xrightarrow{t} y$ as meaning that statement x is "sequentially" followed by y; and $x \xrightarrow{a} y$ as meaning that statement x can "exit" some structure and return to statement y.

A program is a **while** program provided it is an ancestor tree that satisfies: $y \xrightarrow{a} x$ implies $\exists\, y_1, \ldots, y_k$ such that $x \xrightarrow{p} y_1 \xrightarrow{t} y_2 \xrightarrow{t} \cdots \xrightarrow{t} y_k = y$, where y is a leaf and no y_i for $i < k$ has an ancestor pointer (see Figure 6(a)). The last restriction reflects the fact that only the last statement in a **while** loop is allowed to exit the loop.

A program is a **do forever** program provided it is an ancestor tree that satisfies: $y \xrightarrow{a} x$ implies $\exists\, y_1, \ldots, y_k = y$ such that $x \xrightarrow{p} y_1 \xrightarrow{t} y_2 \xrightarrow{t} \cdots \xrightarrow{t} y_k$, where each y_i can have ancestor pointers only to x (see Figure 6(b)). The key distinction between **while** programs and **do forever** programs is that in a **do forever** program all statements in a loop can potentially exit immediately out of the looping structure. Clearly, **do forever** programs correspond to the Ω_n ($n \geq 1$) structures of Böhm and Jacopini [2].

Finally, a label **exit** program is any program that is also an ancestor tree. Essentially label **exit** programs allow any jumping out of substructures as long as the return is always to an ancestor. The class of label **exit** programs is therefore quite extensive and includes many types of so-called "structured" programs (cf. Peterson et al. [15]). For example, all label **exit** programs are *reducible* in the sense of [7]; moreover, they correspond to programs in Engeler normal form [6].

Example. The following program contains label **exit**, **do forever**, and **while** control structures; its representation using the conventions outlined above is shown in Figure 7.

```
L: S1;
     while B1 do
       begin S2;
         do forever
           begin S3; exit L;
             do forever begin S4; exit; S5 end;
             S6
           end;
         S7
       end;
     S8; □
```

3. *S, T Bounded Embeddings*

The following definition is fundamental to what follows. Let $G = (V, E)$ and $G^* = (V^*, E^*)$ be directed graphs with associated metrics d_G and d_{G^*}. We say that G^* can *simulate* G (or G can be *embedded* into G^*) with space constant S and time constant T, written $G \leq_{S,T} G^*$, if there is a mapping (called an *embedding*) $\Phi : V^* \to V \cup \{\Lambda\}$ of the nodes

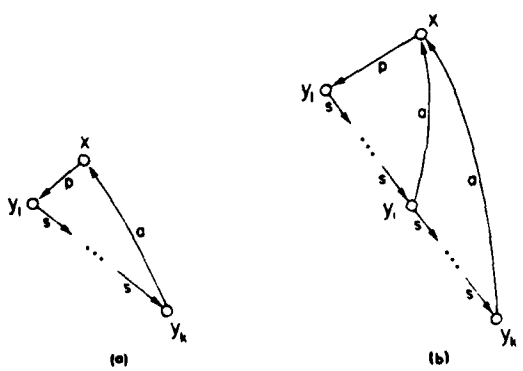


FIG. 6

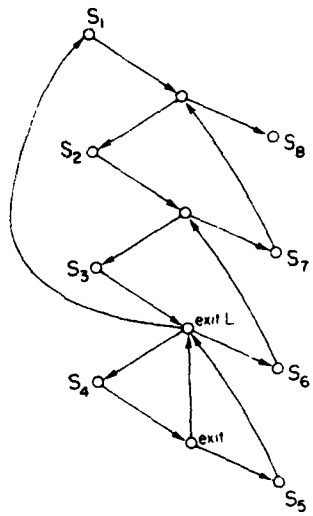


FIG. 7

of G^* to the nodes of G and a special node Λ, so that:

(1) $\forall v^* \in V^*$ with $\Phi(v^*) \neq \Lambda$,
$\forall w \in V$ such that $d_G(\Phi(v^*), w) < \infty$,
$\exists\, w^* \in V^*$ such that $\Phi(w^*) = w$ and $d_G(v^*, w^*) \leq T \cdot d_G(\Phi(v^*), w)$;
(2) $\forall v \in V$, $0 < |\Phi^{-1}(v)| = |\{v^* \in V^* : \Phi(v^*) = v\}| \leq S$.

If Φ is an embedding and $\Phi(v^*) = \Lambda$, then we refer to v^* as a *bookkeeping* node. If $\Phi(v^*) = v \neq \Lambda$, then v^* is said to be a *copy* of v. If $S = 1$, we often write $\leq_T$ instead of $\leq_{1,T}$.

Condition (1) states that when G and G^* are control structures (or data structures) simulation involves at most a T-fold increase in the cost of statement sequencing (or data element accessing), i.e. the embedding induces at most a T-fold increase in path length. Condition (2) states that there are at most S copies of any $v \in V$ in G^*. Note that although $G \leq_{S,T} G^*$ may hold between data structures G and G^* when $S > 1$, it is unlikely that such a simulation would be of value (e.g. if an array is being stored as a list structure with multiple copies of array elements, then selective updating of the array may involve multiple updating of list nodes). For control structures, however, simulations with $S > 1$ are frequently used and are quite natural; this is sometimes called *node splitting*.

Example. Consider the flow diagrams shown in Figure 8. Figure 8(b) is the result of applying a standard "restructuring" algorithm [1] to Figure 8(a) to remove the multiple exit loop x_3, x_4, x_5. Viewing both diagrams as directed graphs, the graph in Figure 8(b) is a 2,2 simulation of Figure 8(a) by defining Φ as follows:

$$\begin{aligned}
&\Phi(x_1^*) = x_1, && \Phi(x_6^*) = x_5,\\
&\Phi(x_2^*) = x_2, && \Phi(x_{11}^*) = x_6,\\
&\Phi(x_3^*) = \Phi(x_7^*) = (x_{10}^*) = \Lambda, && \Phi(x_{12}^*) = \Phi(x_{13}^*) = x_7,\\
&\Phi(x_4^*) = \Phi(x_8^*) = x_3, && \Phi(x_{14}^*) = \Phi(x_{15}^*) = x_8,\\
&\Phi(x_5^*) = \Phi(x_9^*) = x_4, && \Phi(x_{16}^*) = x_9.
\end{aligned}$$

□

4. *Data Structure Embeddings*

In this section we present our main result for data structures, settling negatively the question whether arrays can be stored as arbitrary lists with linear bounds on proximity and determining a nontrivial lower bound on the growth rate of T as a function of n for an $n \times n$ array. This result generalizes a result of Rosenberg [16] showing that arrays

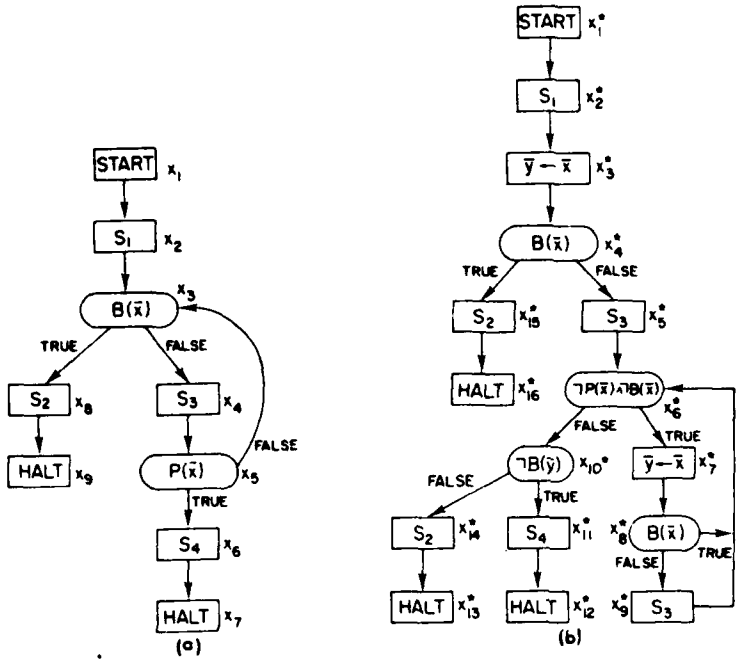


FIG. 8

cannot be stored in linear memory with only bounded loss of proximity. But since the arguments are fundamentally different, it is interesting to compare the two proofs. Recall that Rosenberg's arguments are essentially "volumetric": The number of neighbors within distance n of a node in an array can be quadratic in n, while a node in a linear list can have at most $2n$ such neighbors. A volumetric argument then demonstrates that arrays cannot be stored in a system with such linear neighborhood structure with only bounded loss of proximity. In contrast, such methods do not seem to apply to our problems; e.g. a node in a binary tree can have more than 2^n neighbors within distance n.

To obtain our result we need a series of lemmas. Let $G = (V, E)$ be a directed graph with associated metric d_G and suppose $A \subseteq V$. We define the *boundary* of A as follows:

$$\partial(A) = \{y \in A : \exists\, x \notin A \text{ such that } d_G(x, y) = 1\}.$$

In other words, $\partial(A)$ is the set of nodes in A reachable from some node not in A by an arc of G.

LEMMA 4.1. *Let $G_n = (V_n, E)$ be an $n \times n$ array and suppose that $A \subseteq V_n$ is such that $|A| \leq \frac{1}{2}n^2$. Then $|A| \leq 2|\partial(A)|^2$.*

PROOF. We assume $|A| > 0$, since otherwise the lemma is trivially true, and let $A_1, \ldots, A_n$ be the *columns* of A; that is, if $\{(1, i), (2, i), \ldots, (n, i)\}$ is the ith column of G_n, then A_i is that subset of the column that is included in A. Let k be the number of columns A_i such that $|A_i| < n$, and let $l \leq k$ be the number of columns A_i with $0 < |A_i| < n$. Since $|A| \leq \frac{1}{2}n^2$, it follows that $(n - k)n \leq \frac{1}{2}n^2$ and hence

$$k \geq \tfrac{1}{2}n. \tag{1}$$

Notice that if $0 < |A_i| < n$, then at least one node in A_i is adjacent to a node not in A and thus contributes at least one node to $\partial(A)$; therefore

$$|\partial(A)| \geq l. \tag{2}$$

Suppose that $|A_{i_0}| = 0$ for some i_0, $1 \leq i_0 \leq n$. We then claim

$$| \partial(A) | \geq \max_j | A_j |. \tag{3}$$

To show this, let A_j be maximal in size and assume $i_0 < j$, the case $j < i_0$ being handled symmetrically. Select any row r of G_n such that $(r, j) \in A_j$. Now, $(r, i_0) \notin A$ by assumption, but some one or more of $(r, i_0 + 1), \ldots, (r, j)$ is in A. Therefore each row r of G_n for which $(r, j) \in A_j$ contributes at least one node to $\partial(A)$, which establishes (3).

To complete the proof of the lemma we consider two cases.

I. No A_i is empty. In this case, $l = k$, and by combining (1) and (2),

$$2 | \partial(A) |^2 \geq 2l^2 = 2k^2 \geq \tfrac{1}{2}n^2 \geq | A |.$$

II. Some A_i is empty. Let $c_1, \ldots, c_m$ denote the cardinalities of the nonempty columns A_j. If some $c_p = n$, then the result follows directly from (3). If not, then $m = l$ and $c_1 + \cdots + c_m = | A |$, so that $\max_j | A_j | \geq | A |/l$. By (2) and (3) it follows that $2 | \partial(A) | \geq l + | A |/l$. The lemma is now immediate by calculation. □

LEMMA 4.2. *Let $G_n = (V_n, E)$ and suppose $x, y \in V_n$; then $d_{G_n}(x, y) \leq 2n$.*

PROOF. This is an elementary property of arrays. □

Lemmas 4.1 and 4.2 and the fact that $| V_n | = n^2$ summarize the basic properties of arrays that will be used in the proof of our main result.

LEMMA 4.3. *Let $H = (V, E)$ be an ancestor tree and let $H_0 = (V_0, E_0)$ be a subtree of H. If $x \in V_0$ and $y \in V - V_0$, then $d_H(y, x)$ is greater than or equal to the depth of x in H_0.*

PROOF. Since $y \notin V_0$, any path from y to x must pass through the root of H_0. □

LEMMA 4.4. *Let $H^* = (V^*, E^*)$ be an ancestor tree; let $H_0^* = (V_0^*, E_0^*)$ be a subtree of H^*; and let $A = \Phi(V_0^*) - \{\Lambda\}$. If $G_n \leq_T H^*$ and $| A | \leq \frac{1}{2}n^2$, then $T \geq \frac{1}{2}(\log | A | - 1)$; in other words, $| A | \leq 2^{2T+1}$.*

PROOF. Assume that $| \partial(A) | > 2^T$. Since the root of H_0^* has at most 2^T descendants of depth less than $T + 1$, there is a node $x^* \in V_0^*$ of depth greater than or equal to $T + 1$ in H_0^* such that $\Phi(x^*) \in \partial(A)$. Since $\Phi(x^*) \in \partial(A)$, there is a $y \in V_n - A$ with $d_{G_n}(y, \Phi(x^*)) \leq 1$. Now there exists a y^* such that $\Phi(y^*) = y$ and $d_{T^*}(y^*, x^*) \leq T$, by the definition of $\leq_T$. Since $y \notin A$ it follows that $y^* \notin V_0^*$. But by Lemma 4.3, $d_{H^*}(y^*, x^*) \geq T + 1$, which is a contradiction. Therefore, $| \partial(A) | \leq 2^T$ and by Lemma 4.1 $| A | \leq | \partial(A) |^2 \leq 2^{2T+1}$. □

THEOREM 4.5. *Let $H^* = (V^*, E^*)$ be an ancestor tree. If $G_n \geq_T H^*$, then $T \geq \frac{1}{3}\log n - \frac{2}{3}$.*

PROOF. Assume $G_n \leq_T H^*$ and for any subtree $H_i = (V_i^*, E_i^*)$ of H^* let $A_i = \Phi(V_i^*) - \{\Lambda\}$. Let H_1^* and H_2^* be subtrees of some node in H^*. Either $| A_1 | \leq \frac{1}{2}n^2$ or $| A_2 | \leq \frac{1}{2}n^2$, since Φ is 1-1. Using this fact, we may assume that H^* is of the form shown in Figure 9, where $| A_i | \leq \frac{1}{2}n^2$ for $1 \leq i \leq k$. (We have suppressed explicit representation of ancestral links.) Without loss of generality we assume always that the "smaller" subtree is on the right. By Lemma 4.4, $| A_i | \leq 2^{2T+1}$ for all i.

Let i be the smallest integer such that $| A_i | \neq 0$, and let j be the largest such integer. Then $\sum_{l=i}^{j} | A_l | \leq (j - i + 1)2^{2T+1}$. Since $| V_n | = n^2$,

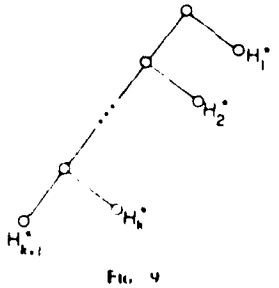


FIG. 9

$$(j - i + 1)2^{2T+1} \geq n^2. \tag{1}$$

Now let $x^* \in V_j^*$ and $y^* \in V_i^*$. Then by Lemma 4.3, $d_{H^*}(y^*, x^*) \geq j - i$.

On the other hand, by Lemma 4.2, $d_{G_n}(\Phi(y^*), \Phi(x^*)) \leq 2n$; hence, since $G_n \leq_T H^*$, $d_{H^*}(y^*, x^*) \leq 2nT$. Thus

$$j - i \leq 2nT. \tag{2}$$

Combining (1) and (2), we have $2nT + 1 \geq n^2/2^{2T+1}$. It follows that $T \geq \frac{1}{2}\log n - \frac{3}{2}$. □

5. *Main Theorem*

Observe that the $S = 1$ hypothesis was used at several key points in the proof of Theorem 4.5. Since this restriction is unrealistic in dealing with control structures, we now remove it by generalizing the previous result.

THEOREM 5.1. *Let $H^* = (V^*, E^*)$ be an ancestor tree and let $G_n \leq_{S,T} H^*$ where G_n is an $n \times n$ array. Then $T + \log S \geq \log n - \log 8\sqrt{2}$.*

PROOF. Let Φ be the embedding function, and define a new function Ψ mapping subsets of V^* to subsets of V by $\Psi(A^*) = \Phi(A^*) - \{\Lambda\}$. In other words, $\Psi(A^*)$ contains those nodes of V for which copies exist in A^*.

As in Theorem 4.5, we decompose H^* as follows. Let x_1^* be the root of H^* and write H^* as in Figure 10(a), where we may assume $|\Psi(L_1^*)| \leq |\Psi(R_1^*)|$ without loss of generality. Clearly, this process can be repeated, letting x_{i+1}^* denote the root of R_i^* and expanding R_i^* at each stage of the construction. Thus, H^* can be written as in Figure 10(b), where $|\Psi(L_i^*)| \leq |\Psi(R_i^*)|$ for $1 \leq i \leq k$. Notice that we have ignored all ancestral links in this construction. Indeed, we assume that all such links exist but suppress explicit reference to them.

Let $H_i^* = (V_i^*, E_i^*)$ denote the subtree of Figure 10(c). We say that H_i^* is *small* if $|\Psi(V_i^*)| \leq \frac{1}{2}n^2$; otherwise H_i^* is *large*. (The notion of smallness is motivated by the key to the argument in Theorem 4.5.) Let

$$D_k = \bigcup_{\substack{1 \leq i \leq k \\ H_i^* \text{ small}}} \Psi(V_i^*).$$

In other words, D_k is the set of nodes in V of which copies exist in some small H_i^*.

LEMMA 5.2. *For some p, $\frac{1}{2}n^2 \leq |D_p| < \frac{1}{2}n^2$.*

PROOF. By convention, $|D_0| = 0$. If $|D_{p-1}| < \frac{1}{2}n^2$ *and* $|D_p| \geq \frac{1}{2}n^2$, then $D_p = D_{p-1} \cup \Psi(V_p^*)$, where H_p^* is small, so that

$$|D_p| \leq |D_{p-1}| + |\Psi(V_p^*)| < \tfrac{1}{2}n^2 + \tfrac{1}{2}n^2 = \tfrac{1}{2}n^2.$$

Thus we need show only that $|D_p| \geq \frac{1}{2}n^2$ for some p.

Since $|\Psi(L_1^*)| \leq |\Psi(R_1^*)|$ and $\Psi(V^*) = \Psi(L_1^*) \cup \Psi(x_1^*) \cup \Psi(R_1^*)$, it follows that $n^2 = |\Psi(V^*)| \leq |\Psi(L_1^*)| + 1 + |\Psi(R_1^*)| \leq 2|\Psi(R_1^*)| + 1$; hence $|\Psi(R_1^*)| \geq \frac{1}{2}(n^2 - 1) \geq \frac{1}{3}n^2$. We can obviously choose p so large that $|\Psi(R_p^*)| = 0$. We claim that the associated D_p is large.

Let i be the largest integer such that $|\Psi(R_i^*)| \geq \frac{1}{3}n^2$. Since $|\Psi(R_1^*)| \geq \frac{1}{3}n^2$, such an i

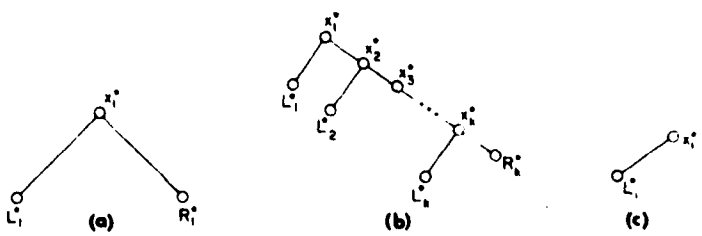


FIG 10

must exist. Then $|\Psi(R_i^*)| < \frac{1}{4}n^2$ for $i < j \le p$, and since $\Psi(V_j^*) = \Psi(L_j^*) \cup \Psi(x_j^*)$, we have

$$|\Psi(V_j^*)| \le 1 + |\Psi(L_j^*)| \le 1 + |\Psi(R_j^*)| < 1 + \tfrac{1}{4}n^2.$$

Thus H_j^* is small for $i < j \le p$. But this implies $\Psi(R_i^*) \subseteq \bigcup_{i<j\le p} \Psi(V_j^*) \subseteq D_p$. By our choice of i, however, we conclude that $|D_p| \ge |\Psi(R_i^*)| \ge \frac{1}{4}n^2$, establishing our claim. □

We now introduce a variant of the concept of boundary. If A is a set of nodes of G_n, then the *coboundary* of A is defined by

$$\tilde{\partial}(A) = \{y \notin A: \text{there exists } x \in A \text{ such that } d_{G_n}(x, y) = 1\}$$
$$= \partial(V_n - A).$$

In other words, $\tilde{\partial}(A)$ is the set of nodes not in A reachable from some node in A in one step. The proof of the following result is similar to that of Lemma 4.1 and is omitted.

LEMMA 5.3. *Let A be a set of nodes of G_n with $|A| \le \frac{1}{4}n^2$. Then $|A| \le 2|\tilde{\partial}(A)|^2$.*

Let k satisfy Lemma 5.2. By Lemma 5.3, $|\tilde{\partial}(D_k)| \ge |D_k|^{1/2}/\sqrt{2} \ge n/2\sqrt{2}$. Now let $l = |\{H_i^* : H_i^* \text{ is large}\}|$, the number of large subtrees. Since at most S copies of any node in G_n appear in H^*,

$$ln^2/4 \le \sum_{\substack{1\le i\le k \\ H_i^* \text{ large}}} |\Psi(V_i^*)| \le Sn^2.$$

Hence $l \le 4S$.

In order to complete the proof we proceed as follows. We have already shown that $|\tilde{\partial}(D_k)| \ge n/2\sqrt{2}$; we show next that this implies that there are too many paths into the large trees H_i^* from the small trees for S and T to be bounded.

Let

$$Q_T = \{v^* \in V_i^* : H_i^* \text{ is large and there exist } H_j^* \text{ small and } x^* \in V_j^* \text{ such that } d_{H^*}(x^*, v^*) \le T\}.$$

In other words, Q_T is the set of nodes in large subtrees H_i^* that can be reached from some node in some small subtree H_j^* in at most T steps. We define a one-to-one mapping g from $\tilde{\partial}(D_k)$ into Q_T as follows. Select some $y \in \tilde{\partial}(D_k)$. Then $y \notin D_k$ and, for some $x \in D_k$, $d_{G_n}(x, y) \le 1$. Let x^* be a copy of x in some small H_i^*. Such a copy exists by the definition of D_k. Since $G_{n\le S,T}\, H^*$, there is a copy y^* of y such that $d_{H^*}(x^*, y^*) \le T$. Now y^* is not in any small H_i^* since $y \notin D_k$. Thus we can define $g(y) = y^*$, and g is indeed a mapping from $\tilde{\partial}(D_k)$ to Q_T. In order to see that g is one-one, we note that for any $y \in \tilde{\partial}(D_k)$, $\Phi g(y) = \Phi(y^*) = y$; hence g is one-one, so that $|Q_T| \ge |\tilde{\partial}(D_k)|$.

Thus we have, on the one hand, that $|Q_T| \ge |\tilde{\partial}(D_k)| \ge n/2\sqrt{2}$ and, on the other hand, that

$$|Q_T| \le |\{H_i^* : H_i^* \text{ is large}\}| \cdot |\{v^* : v^* \in \text{large } H_i^* \text{ within depth } T \text{ of the root of } H_i^*\}|$$
$$\le l \cdot 2^T \le 4S \cdot 2^T.$$

Combining the upper and lower bounds on $|Q_T|$, we deduce that $T + \log S \ge \log n - \log 8\sqrt{2}$. □

As an application of Theorem 5.1, we present the following result. Informally a flowchart is said to be in Engeler normal form if it is represented by a tree augmented by pointers from nodes to ancestors, or nodes at an earlier level but along the same branch. More precisely, a **go to** program G has an S, T Engeler normal form if $G \le_{S,T} H$ for some ancestor tree H.

COROLLARY 5.4. (1) *If G_n has an S, T Engeler normal form and T is fixed, then $S \ge c \cdot n$.* (2) *If G_n has an S, T Engeler normal form and S is fixed, then $T \ge c \cdot \log n$.*

Thus in the worst case, either time or space must be unbounded in the construction of Engeler normal forms.

6. *Control Structures*

In this section we establish our main results for control structures, using the relation $\leq_{S,T}$ (see Figure 1). For classes X and Y of control structures, i.e. classes of graph representations of programs constructed using only control structures from the indicated restricted class of control structures, we say that X is *more powerful* than Y when there exist constants S', T' such that (1) for all $H \in Y$ there exist $G \in X$ such that $H \leq_{S',T'} G$, but for no constants S, T is it true that (2) for all $G \in X$ there exists $H \in Y$ such that $G \leq_{S,T} H$.

Since for the hierarchy of Figure 1 if X is more powerful than Y, then the control structures in Y are restrictions of the control structures in X, condition (1) is trivially satisfied with $S' = T' = 1$. It is, of course, the results that establish condition (2) that have the greatest novelty.

To place our results in historical perspective, we follow Ledgard [12] in distinguishing the following extremes in simulations among control structures:

(1) G is *functionally* simulated by H (written $G \leq_f H$) if, under identical interpretations, G and H compute the same function.

(2) G is *very strongly* simulated by H (written $G \leq_{vs} H$) if $G \leq_{1,1} H$ and if Φ is an embedding inducing $\leq_{1,1}$, then the domain of Φ and the range of Φ are *identical* sets.

In [2] it is shown that for each **go to** program G there exists a **while** program H such that $G \leq_f H$, while in [10] it is shown that for some **go to** program G there does not exist a **while** program H such that $G \leq_{vs} H$. Several other notions of simulation intermediate to $\leq_f$ and $\leq_{vs}$ have also been used to study the relative power of classes of control structures [1, 3, 11, 15].

The connection between our relation $\leq_{S,T}$ and these relations is:

(1) $\leq_{S,T}$ is weaker than $\leq_{vs}$, since we allow both space and time to increase and do not require Φ to have identical range and domain;

(2) $\leq_{S,T}$ is stronger than $\leq_f$, since we require that paths be preserved in a weak sense;

(3) $\leq_{S,T}$ deals only with combinatorial aspects of program structure, and thus we make no assumptions about adding program variables or extra predicates (as were made for example in [1, 2, 11]).

We thus claim that the hierarchy theorems presented in this section span the relations used in previous studies. For the remainder of this section we adopt the notation $X \not\leq Y$ to indicate that $Y \subset X$ but the graphs in X are not uniformly simulated by the graphs in Y, i.e. X is more powerful than Y.

We will make use of the following definitions. For any directed graph G let

$$N^G_{\text{in}}(l, x) = |\{y : d_G(y, x) \leq l\}| \qquad N^G_{\text{out}}(l, x) = |\{y : d_G(x, y) \leq l\}|.$$

LEMMA 6.1. *Suppose that $G \leq_{S,T} G^*$ and let x be a node in G. Then (1) for any copy x^* of x, $N^G_{\text{out}}(l, x) \leq N^{G^*}_{\text{out}}(Tl, x^*)$; and (2) for some copy x^* of x, $N^G_{\text{in}}(1, x) \leq S \cdot N^{G^*}_{\text{in}}(T, x^*)$*

The proofs of both (1) and (2) follow easily from the definition of $\leq_{S,T}$ and are left to the reader.

THEOREM 6.2. **do forever $\not\leq$ while.**

PROOF. Let S, T be such that for all **do forever** programs G, there is a **while** program H for which $G \leq_{S,T} H$. By part (2) of Lemma 6.1, for any node x in G there exists a copy x^* of x such that $N^G_{\text{in}}(1, x) \leq S \cdot N^H_{\text{in}}(T, x^*)$. But since H is a **while** program, nodes in H have at most one ancestor pointer to them, so that $N^H_{\text{in}}(T, x^*) \leq 2^T$. Thus $N^G_{\text{in}}(1, x) \leq S2^T$ for any **do forever** program G and any node x in G. This is a contradiction, since the number of ancestor pointers to nodes in **do forever** programs can be unbounded. □

THEOREM 6.3. *label* **exit $\not\leq$ do forever.**

PROOF. Let S, T be such that for any label **exit** program G there exists a **do forever** program H such that $G \leq_{S,T} H$. Consider the label **exit** graph $G^{(n)}$ defined as follows:

(1) $V^{(n)} = \{x_1, \ldots, x_n\}$, (2) $x_i \xrightarrow{e} x_{i+1}$ for all $1 \leq i < n$, (3) $x_n \xrightarrow{e} x_1$ for all $1 \leq i < n$.

(See Figure 11.) Then, by construction, $N^{G^{(n)}}_{\text{out}}(1, x_n) = n - 1$.

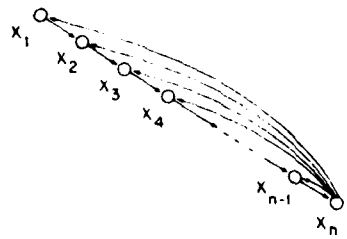


FIG. 11

Let $H^{(n)}$ be the corresponding **do forever** program. Then a node x^* in H^n has at most two sons and one ancestor, so that $N_{\text{out}}^{H^{(n)}}(T, x^*) \leq 3^T$. Thus by part (1) of Lemma 6.1, $n - 1 = N_{\text{out}}^{G_n}(1, x_n) \leq N_{\text{out}}^{H^n}(T, x^*) \leq 3^T$, where x^* is any copy of x_n in H^n, a contradiction. □

THEOREM 6.4. *For $d \geq 2$,* **go to**$_d \not\leq$ *label* **exit**.

PROOF. Notice that arrays are included in **go to**$_f$ for $f \geq 4$. Thus, by Theorem 5.1, we have **go to**$_f \not\leq$ label **exit** ($f \geq 4$), since label **exit** programs are ancestor trees. To complete the proof it is sufficient to note that for any array G_n there is some H in **go to**$_2$ such that $G_n \leq_{1,4} H$. □

THEOREM 6.5. **go to**$_\omega \not\leq$ **go to**$_d$ *for all* $d \geq 1$.

PROOF. Let S, T be such that for all **go to**$_\omega$ programs G, there exists a **go to**$_d$ program H such that $G \leq_{S,T} H$. By part (1) of Lemma 6.1, for all nodes x in G there is a copy x^* of x in H such that $N_{\text{out}}^H(T, x^*) \geq N_{\text{out}}^G(1, x)$. But $N_{\text{out}}^H(T, x^*) \leq d^T$, since the out-degree of any vertex in H is at most d. This is a contradiction, since *in* **go to**$_\omega$ $N_{\text{out}}^G(1, x)$ is unbounded. □

7. *Conclusion*

The methods for comparing data structures and control structures by the relation $\leq_{S,T}$ appear to be quite general, and there are several straightforward extensions of the data structure embedding results that recover relationships between other data structures.

In the case of control structures, the conclusions to be drawn are perhaps more widely varying and seem to give direction to further investigations. The observation in Corollary 5.4 that a standard schema construction is inherently inefficient leads us to question the status of other property preserving transformations. We also believe that Theorem 6.5 has implications for the often quoted "theoretical foundations of structured programming"; this is particularly apparent because the **go to** program that enters the proof is itself a highly structured object. Indeed, a $\leq_{S,T}$ embedding into a label **exit** program fails, not because G_n is ill structured, but rather because of the densely hierarchical nature of the control flow. We thus offer programs of the form G_n as "structured" **go to** programs whose structures cannot be maintained by less general control structures. The exact relationship between ancestor trees and reducible flow graphs [7] is still unsettled, but some work has been done to place the reducible flow graphs in the hierarchy of Figure 1 [5]. More recent extensions of the results presented here give techniques for uncovering *total* space and time simulation trade-offs, as opposed to the worst-case analyses of this paper [5]. Finally, the extension of these results to parallel and asynchronous control structures appears to be possible and promises to yield important information about the relative power of nonsequential mechanisms.

ACKNOWLEDGMENTS. The authors would like to thank W. Wesley Peterson for his comments on an earlier version of this paper and Arnold Rosenberg for a careful reading of the paper that led to a material improvement in presentation and to improvements to several proofs.

REFERENCES

1. ASHCROFT, E., AND MANNA, Z. The translation of GOTO programs to WHILE programs. Proc. IFIP Cong. 1971, North-Holland Pub. Co., Amsterdam, 1972, pp. 250–255.
2. BÖHM, C., AND JACOPINI, G. Flow-diagrams, Turing machines, and languages with only two formation rules. *Comm. ACM* 9, 3 (March 1966), 366–371.
3. BRUNO, J., AND STEIGLITZ, K. The expression of algorithms by charts. *J. ACM 19*, 3 (July 1972), 517–525.
4. DAHL, O.-J., ET AL. *Structured programming*. Academic Press, New York, 1972.
5. DEMILLO, R.A., EISENSTAT, S.C., AND LIPTON, R.J. Space-time tradeoffs in structured programming (to appear).
6. ENGELER, E. Structure and meaning of elementary programs. *Lecture Notes in Mathematics, No. 188: Symp. on Semantics of Algorithmic Languages*. E. Engeler, Ed., Springer-Verlag, Berlin, 1971, pp. 89–101.
7. HECHT, M.S., AND ULLMAN, J. D. Characterizations of reducible flow graphs. *J. ACM 21*, 3 (July 1974), 367–375.
8. KNUTH, D.E. *The Art of Computer Programming, Vol. 1: Fundamental Algorithms*. Addison-Wesley, Reading, Mass., 1968.
9. KNUTH, D.E. Structured programming with GOTO statements. Rep. STAN-CS-74-416, Computer Sci. Dep., Stanford U., Stanford, Calif., 1974.
10. KNUTH, D.E., AND FLOYD, R.W. Notes on avoiding "go to" statements. *Infor. Process. Lett. 1* (1971), 23–31.
11. KOSARAJU, S.R. Analysis of structured programs. *J. Computer and Syst. Sci. 9*, 3 (June 1974), 232–255.
12. LEDGARD, H.F. A geneology of control structures. Research Rep., Computer and Information Science Dep., U. of Massachusetts, Amherst, Mass., 1974.
13. MOLER, C.B. Matrix computations with FORTRAN and paging. *Comm. ACM 15*, 4 (April 1972), 268–270.
14. PERLIS, A.J., AND THORNTON, C. Symbol manipulation by threaded lists. *Comm. ACM 3*, 4 (April 1960), 201–202.
15. PETERSON, W.W., KASAMI, T., AND TOKURA, N. On the capabilities of the while, repeat, and exit statements. *Comm. ACM 16*, 8 (Aug. 1973), 503–512.
16. ROSENBERG, A.L. Preserving proximity in arrays. Rep. RC-4875, IBM Thomas J. Watson Research Center, Yorktown Heights, N.Y., 1974.

RECEIVED APRIL 1975; REVISED MARCH 1975

SPACE-TIME TRADEOFFS IN STRUCTURED PROGRAMMING: AN IMPROVED COMBINATORIAL EMBEDDING THEOREM

Richard A. DeMillo*
Stanley C. Eisenstat†
Richard J. Lipton†

* School of Information and Computer Science
Georgia Institute of Technology
Atlanta, GA 30332

† Computer Science Department
Yale University
New Haven, CT 06520

These results were announced at the 1976 Johns Hopkins Conference on Information Sciences and Systems. This research was supported in part by the U.S. Army Research Office, Grant Nos. DAHC04-74-G-0179 and DAAG29-76-G-0338; the Office of Naval Research, Grant No. N00014-67-097-0016; and the National Science Foundation, Grant No. DCR-74-12870.

Abstract: Let G and G* be programs represented by directed graphs. We define a relation $\leq_{S,T}$ between G and G* that formalizes the notion of G* simulating G with S-fold loss of space efficiency and T-fold loss of time efficiency, and prove that if $G \leq_{S,T} G^*$, where G has n statements and G* is structured, then in the worst case $T + \log_2\log_2 S \geq \log_2 n + O(\log_2\log_2 n)$.

Keywords and Phrases: ancestor tree, complexity, control structure, directed graph, embedding

CR Categories: 4.22, 4.34, 5.24, 5.32

1. Introduction

In a previous paper [1], we made precise some intuitive observations concerning the efficiency of structured programs by defining a combinatorial relation that corresponds to the notion of *uniform simulation* between programs. Informally, we say that a program G* uniformly simulates a program G if G* carries out the computation of G (and possibly additional computation which might be regarded as "bookkeeping") in such a way that the space-time efficiency of G is degraded by a factor that is independent of the size of G. The main results of [1] indicate that the non-existence of uniform simulations among many well-known classes of control structures is due to the combinatorial aspects of program structure and is not at all related to such details of program organization as choice of data structures or limitations on the form of Boolean expressions.

Indeed, the main result of [1] (Theorem 5.1) provides a non-trivial lower bound on the loss of space-time efficiency in *any* structured simulation of a goto program. This short note extends that result, improving the space-time inequality of [1, Theorem 5.1] by an exponential. Thus we now show that there are goto programs with n statements such that, for any structured simulation, either:

1) the simulation runs at least[†]

$$c_1 \log_2 n$$

times as slow as the original program,

or

2) the simulation has at least $2^{c_2 n^{c_3}}$ statements.

[†] We use c_1, c_2, c_3 to denote positive constants.

I.e., there are goto programs that can only be simulated by either *very slow* or *very large* structured programs.

In the sequel, we will concentrate on the combinatorial theorem that achieves these bounds. The programming language significance of the graphs and relations studied here is discussed extensively in [1].

2. Preliminaries

A *directed graph* G is an ordered pair (V,E) of *vertices* V and *edges* $E \subseteq V \times V$. A *path* in G is an ordered sequence of vertices connected by edges. For vertices $x,y \in V$, let $d_G(x,y)$ denote the length of a minimum length path form x to y. If no such path exists, then $d_G(x,y) = \infty$.

A *binary tree* is a directed graph that consists of either a single vertex or a *root* x and edges between x and the root of each of two binary trees called the left and right *subtrees* of x. A vertex x in a binary tree is a *leaf* if it has no sons. If H = (V,E) is a binary tree with *root* $r \in V$ and leaf $\ell \subset V$, and $P = (x_1,\ldots,x_n)$ is a direct path from $x_1 = r$ to $x_n = \ell$, then P is called a *branch* of H. An *ancestor tree* G = (V,E) is a directed graph with the following properties:

1) There exists a subset $E_o \subseteq E$ such that $G_o = (V,E_o)$ is a binary tree;

2) If $(x,y) \in E - E_o$, then y is an ancestor of x in G_o.

Let G_n denote the $n \times n$ rook-connected array of vertices. If the vertices of G_n are indexed by (i,j) for $1 \le i,j \le n$, then, except for the obvious extremal conventions, there are symmetric edges between (i,j) and (i,j+1), (i+1,j).

For any directed graph G = (V,E), the notion of *boundary* makes sense. Let $A \subseteq V$. Then the *boundary of* A is defined as

$$\partial(A) = \{y \in V-A: \exists x \in A \text{ such that } (x,y) \in E\}$$

Clearly, $\partial(A)$ denotes the set of vertices not in A which are reachable from A by

a single edge.†

By a simple improvement of a result from [1], we have the following important property of arrays:

Lemma 1: (Boundary Lemma) Let A be a set of vertices of G_n with $|A| \leq n^2/2$. Then

$$2|A| \leq |\partial(A)|^2.$$

3. Graph Embedding

The following relation was defined in [1]. Let $G = (V,E)$ and $G^* = (V^*,E^*)$ be directed graphs, and let $S,T > 0$. Then $G \leq_{S,T} G^*$ if there is a partial function (called an *embedding*) $\phi: V^* \to V \cup \{\Lambda\}$, of the nodes of G^* to the nodes of G and a special node Λ, such that

1) $0 \leq |\phi^{-1}(x)| \leq S$ for all $x \in V$;
2) For all $x^* \in \phi^{-1}(V)$, if $d_{G^*}(\phi(x^*),y) < \infty$ for some $y \in V$, then there exists $y^* \in \phi^{-1}(y)$ such that $d_{G^*}(x^*,y^*) \leq d_G(\phi(x^*),y)$.

If $\phi(v^*) = \Lambda$, then we refer to v^* as a *bookkeeping node*. If $\phi(v^*) = v \neq \Lambda$, then v^* is said to be a *copy* of v. Condition (1) states that there are at most S copies of any $v \in V$ in G^*. Condition (2) states that the embedding induces at most a T-fold increase in path length.

Theorem 1: [1, Theorem 5.2] If $S(n)$, $T(n)$ are such that $G_n \leq_{S(n),T(n)} G^*$ for some ancestor tree G^*, then

$$T(n) + \log_2 S(n) \geq \log_2 n + c_1. \tag{1}$$

The right hand side of inequality (1) cannot be improved, since with $S(n) \equiv 1$, the construction of [2] shows that

$$T(n) = O(\log_2 n)$$

† The notion of boundary used here corresponds to the coboundary of [1].

is achievable for any n vertex graph. Theorem 1, however, gives only a linear bound on S(n), and it has been conjectured that a non-polynomial lower bound on S(n) exists. In the next section we obtain such a bound.

4. Main Theorem

In this section, we obtain the following improvement of Theorem 1:

Theorem 2: If G^* is an ancestor tree and $G_n \leq_{S(n),T(n)} G^*$, then

$$T(n) + \log_2\log_2 S(n) \geq \log_2 n - O(\log_2\log_2 n).$$

Proof: For notational convenience, let us systematically confuse a graph with its set of vertices, so that "$x \in G$" and "$x \in V$" mean the same thing if $G = (V,E)$.

We assume $G_n \leq_{S,T} G^*$ via an embedding Φ. For any $A^* \subseteq G^*$, we use $\Phi(A^*)$ to denote the set of $x \in G_n$ which are Φ-images of some $x^* \in A^*$. Henceforth, we assume that G^* is a *binary tree*; it will be obvious as we progress that if G^* contains *ancestor edges*, then the proof is completely unaffected.

Let $P = (x_1^*,\ldots, x_k^*)$ be a path of G^*. Then P is an *admissible path* if it is constructed as follows: For each x_i^* $(1\leq i\leq k)$, let L_i^* denote the subtree of x_i^* containing x_{i+1}^*, and let R_i^* denote the other subtree of x_i^*; then either

a) $\Phi(R_i^*) \geq \Phi(L_i^*)$

or

b) $\Phi(R_i^*) \geq n^2/4$.

Note that the definition of admissible path is more general than that used in [1]. Indeed, it is by proving the existence of many such admissible paths that we obtain our result.

We fix an arbitrary admissible path $P = (x_1^*,\ldots,x_k^*)$ and define for $i = 1,\ldots,k$ the subtree $H_i^* = L_i^* \cup \{x_i^*\}$. We shall say that H_i^* is *small* if $|\Phi(H_i^*)| \leq n^2/4$; otherwise H_i^* is said to be *large*. Let

$$D_j = \bigcup_{\substack{1 \le i \le j \\ H_i \text{ is small}}} \phi(H_i^*);$$

in particular, D_k is the set of vertices in G_n which have copies in some small H_i^*.

Lemma 3: For some j,

$$\frac{n^2}{4} \le |D_j| \le \frac{n^2}{2}.$$

Proof: We need only show that there exists an integer j such that $|D_j| \ge n^2/4$, since if j is the least such integer, then (assuming $|D_o| = 0$)

$$|D_j| \le |D_{j-1}| + |\phi(H_j^*)| < \frac{n^2}{4} + \frac{n^2}{4} = n^2/2.$$

We claim that $|\phi(R_1^*)| \ge n^2/4$. For suppose otherwise, whence $|\phi(L_1^*)| \le |\phi(R_1^*)|$ by the definition of an admissible path. Now

$$\phi(G^*) = \phi(H_1^*) \cup \phi(R_1^*),$$

so that

$$n^2 = |\phi(G^*)| \le |\phi(L_1^*)| + 1 + |\phi(R_1^*)| \le 2|\phi(R_1^*)| + 1,$$

and thus

$$|\phi(R_1^*)| \ge n^2/4.$$

Let j be such that $|\phi(R_j^*)| = 0$, and let i be the largest integer such that $|\phi(R_i^*)| \ge n^2/4$. Then

$$|\phi(R_\ell^*)| < n^2/4, \text{ for } \ell = i+1,\ldots,j.$$

Hence,

$$|\phi(H_\ell^*)| \le 1 + |\phi(L_\ell^*)| \le 1 + |\phi(R_\ell^*)| < 1 + n^2/4 \text{ for all } \ell = i+1,\ldots,n.$$

But then each such H_ℓ^* is small, and therefore

$$\phi(R_i^*) \subseteq \bigcup_{i \le \ell \le j} \phi(H_\ell^*) \subseteq D_j.$$

But by the definition of i, $|D_j| \ge n^2/4$. □

Letting k satisfy Lemma 3, we find that D_k satisfies the hypothesis of the Boundary Lemma, so that

$$|\partial(D_k)| \geq \sqrt{2}|D_k|^{1/2} \geq \frac{n}{\sqrt{2}}$$

Lemma 4: If ℓ_P is the number of large trees H_i^* along an admissible path P, then

$$\frac{n}{\sqrt{2}} \leq \ell_P \, 2^T.$$

Proof: Let

$$Q_T = \{v^* \in H_i^*, \text{ large: for some small } H_j^* \text{ and } x^* \in H_j^*, \; d_G(x^*,v^*) \leq T\}.$$

i.e., Q_T is the set of vertices in large H_i^* which are reachable from some node in a small H_j^* by a path of length at most T. We show that $|\partial(D_k)| \leq |Q_T|$ by defining an injection $g : \partial(D_k) \to Q_T$. For $y \in \partial(D_k)$, choose some $x \in D_k$ adjacent to y. Let x^* be a copy of x in a small H_j^*, let y^* be a copy of y such that $d_{G^*}(x^*,y^*) \leq T$, and set $g(y) = y^*$. Since $\phi g(y) = \phi(y^*) = y$, g is one-one. Thus, from (2),

$$|Q_T| \geq |\partial(D_k)| \geq \frac{n}{\sqrt{2}},$$

but

$$|Q_T| \leq |\{H_i^* : H_i^* \text{ large}\}| \cdot |\{v^* : v^* \in H_i^*, \text{ large; } v^* \text{ within distance T of root of } H_i^*\}|$$
$$\leq \ell_P \cdot 2^T \qquad \square$$

To complete the proof, we now show that there are at least $2^{n/2^{T+1/2}}$ admissible paths. Since each admissible path corresponds to a distinct leaf[†] of G^* and $G_n \leq_{S,T} G^*$, we have

$$2^{\frac{n}{\sqrt{2}} 2^{-T}} \leq |\phi^{-1}(V)| \leq S|V| = Sn^2$$

and the result follows.

[†] Without loss of generality, we assume that no leaf of G^* is a bookkeeping node.

Lemma 5: There exist at least $2^{\ell_{min}}$ admissible paths, where $\ell_{min} = \frac{n}{\sqrt{2}} \cdot 2^{-T}$.

Proof: We prove the result by showing that at least ℓ_{min} independent binary choices must be made to construct an arbitrary admissible path. Consider a partial admissible path $x_1,\ldots,x_k$ (i.e., the initial segment of an admissible path). If only one subtree of x_k is large, then the admissible path can only be extended down that subtree. However, if both subtrees are large, then the admissible path can be extended down either subtree without violating the condition (a-b). By Lemma 4, there are at least ℓ_{min} large subtrees along every admissible path, and, for each such subtree, there is a node in the admissible path with two large subtrees. □□

By using the modeling strategy detailed in [1], we obtain the following:

Corollary: For each n there is an n statement goto program Q such that for *any structured simulation* of Q either

1) the simulating program is slower than Q by a factor of $c_1 \log n$, or

or

2) the simulating program is larger than Q by a factor of $2^{c_2 n^{c_3}}$.

An interesting interpretation of this result as a space-time tradeoff is shown in Figure 1, which illustrates, for fixed $n > 0$,

$$S(T,n) \geq 2^{n/2}$$

For any fixed value $K \leq T \leq c_1 \log n$, limiting the loss of time efficiency in the simulating program, the shaded region of Figure 1 shows the only values of S,T which are achievable.

Acknowledgements: We would like to thank Nancy Lynch, Ronald Rivest, Albert Meyer and Arnold Rosenberg for suggesting that we look for the improved embedding theorem contained in this paper.

References

1. R. J. Lipton, S. C. Eisenstat, and R. A. DeMillo, "Space-Time Hierarchies for Control Structures and Data Structures," *Journal of the ACM,* Vol. 23, No. 4, October 1976, pp. 720-737.

2. R. A. DeMillo, S. C. Eisenstat, and R. J. Lipton, "Preserving Average Proximity in Arrays," *Communications of the ACM,* Vol. 21, No. 3, March 1978, pp. 228-231.

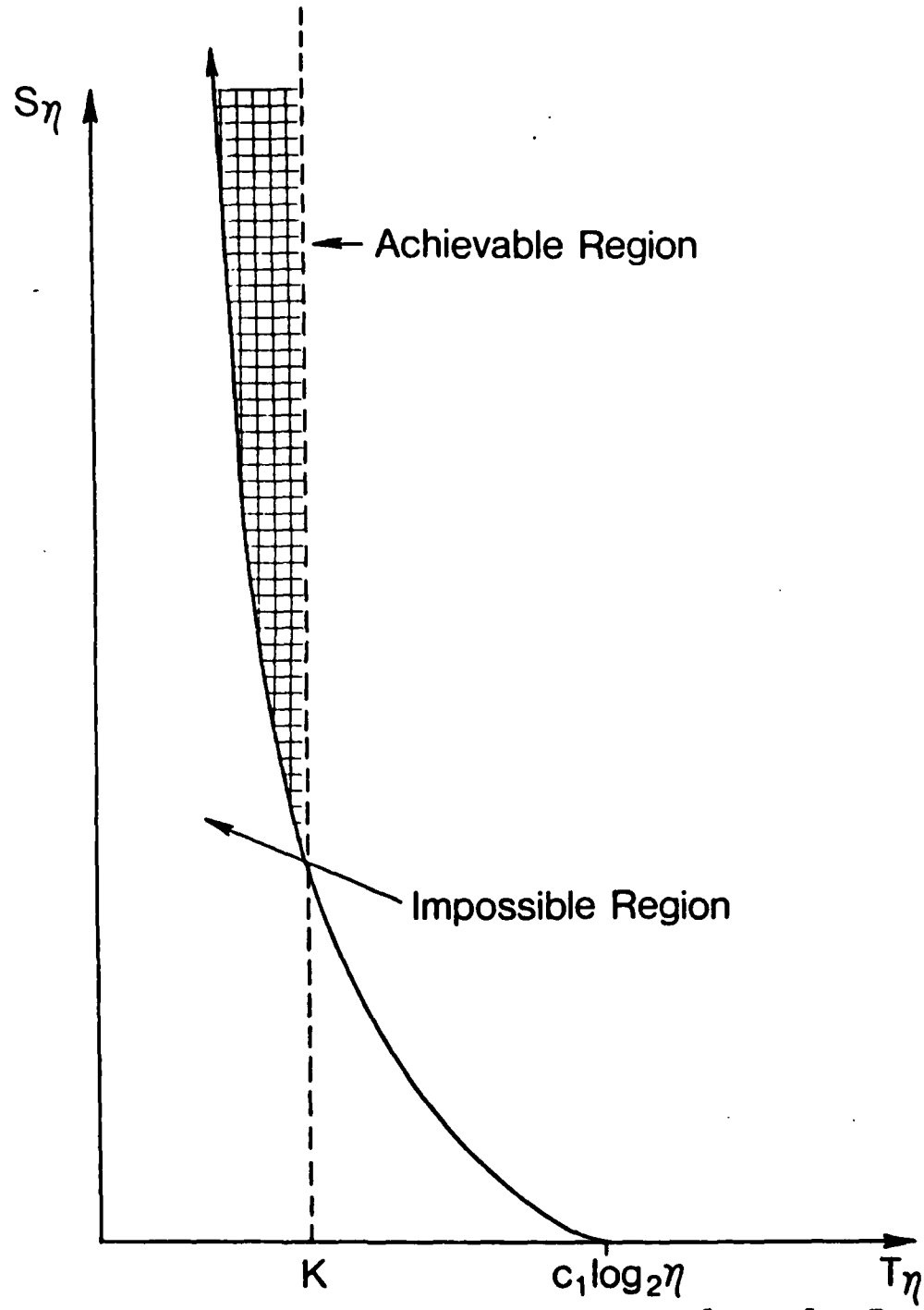


Figure 1. Trading-off T_η for $S_\eta = \left[2^{\eta/2.5}\right], 2^{-T_\eta}$ for fixed program size η.

AN EMBEDDING RESULT FOR LABELLED PROGRAMS

Richard A. DeMillo*

School of Information and Computer Science
Georgia Institute of Technology
Atlanta, GA 30332

S. Rao Kosaraju**

Department of Electrical Engineering
Johns Hopkins University
Baltimore, MD 21218

* Supported in part by the U.S. Army Research Office, Grant Nos. DAAG29-76-G-0338 and DAHC04-74-G-0179.

** Supported in part by the National Science Foundation, Grant No. DCR75-09904.

INTRODUCTION

There are two natural methods of limiting the use of labels in structured programs: bounding the number of labels that can be referenced by a single statement and bounding the total number of labels which can appear in a program. It is implicit in an argument of [1] in the former case and in the unbounded analog of both cases that a genuine limitation *is* imposed and power increases with the number of labels. We show here that, in the latter case, programs with differing bounded numbers of labels are provably equivalent in the precise sense of [1,2]. From [3] it is known that suitable restrictions on the notion of equivalence result in provable differences among these constructs; these restrictions, however, rely on the details of program organization. Hereafter, we deal only with combinatorial arguments. Further motivation for the combinatorial properties in the sequel may be found in [1].

PRELIMINARIES

A directed graph G is composed of a set of *vertices*, V(G), and *arcs* $E(G) \subset V(G) \times V(G)$. The arcs (x,y) and (y,x) together form an *edge* of G. Arcs and edges are represented by directed and undirected arrows, respectively. A *path* from x to y is a sequence of arcs

$$(x, x_1), (x_1, x_2), \ldots, (x_{n-2}, x_{n-1}), (x_{n-1}, y),$$

and such a path is said to be of *length n*. We define the distance metric $d_G(x,y)$ to be the minimum of the lengths of all paths from x to y.

A *binary tree* with root x is a directed graph that is either a single vertex x or contains a vertex x connected by *edges* to root(s) y_i of subtree(s) G , $i \leq 2$. Note that d_G is symmetric on binary trees. Let G be a binary tree with root x_0 and consider the path $(x_0, x_1), \ldots, (x_{n-1}, x_n)$, where $x_i \neq x_j$ for $i \neq j$. We define the following relations on G:

(1) x_j is a *descendent* of x_i $(0 \leq i < j \leq n)$

(2) x_i is an *ancestor* of x_j $(0 \leq i < j \leq n)$

(3) x_i is the *father* of x_{i+1} $(0 \leq i < n)$, and we write $x_i = f_G(x_{i+1})$

(4) x_{i+1} is a *son* of x_i $(0 \leq i < n)$

(5) x_n is *leaf* of G if $f_G(y) \neq x_n$ for all $y \in V(G)$

In a binary tree G, the subtree with root x is denoted by G_x.

An *ancestor tree is a directed graph G whose arcs may be partitioned into two* maximal subsets E_1, E_2 such that $(V(G), E_1)$ is a binary tree and if $(x,y) \in E_2$, then y is an ancestor of x in the binary tree $(V(G), E_1)$. Thus, in an ancestor tree a vertex may be connected by an arc to any of its ancestors. We use the special notation $x \overset{a}{\rightarrow} y$, if $(x,y) \in E_2$. The following terminology is suggested by [1]: if $y \overset{a}{\rightarrow} x$ then x is a *label* and y is an *exit*.

We then say that an ancestor tree, G, is a *k-label program* if it contains at most k-labels; we also say that G is a *k-exit program* if it contains at most k-exits.

SPACE-TIME BOUNDED SIMULATIONS

The following definition is from [1]; it introduces a fundamental mechanism for comparing programs. Let G, G* be directed graphs. We say that G* *simulates* G with *space* dilation S > 0 and *time* dilation T > 0, written $G \leq_{S,T} G^*$ if there is a map (called an *embedding* of G in G*).

$$\Phi: V(G^*) \to V(G) \cup \{\Lambda\}, \ \Lambda \notin V(G) \cup V(G^*)$$

such that:

(1) $\forall\, u \in V(G)$

$$0 < |\, \Phi^{-1}(u)| \leq S \ ,$$

and

(2) $\forall\, v^* \in V(G^*)$ such that $\Phi(v^*) \neq \Lambda$

$\forall\, w \in V(G)$ such that

$$d_G(\Phi(v^*), w) < \infty$$

$\exists\, w^* \in V(G^*)$ such that $\Phi(w^*) = w$, and

$$d_{G^*}(v^*, w^*) \leq T \cdot d_G(\Phi(v^*), w) \ .$$

If Φ is an embedding and $\Phi(u^*) = \Lambda$, then u^* is said to be a *bookkeeping* vertex; on the other hand, if $\Phi(u^*) = u \neq \Lambda$, then u^* is said to be a *copy* of u. Clearly, in a simulation of G with space dilation S, no vertex of G can have more than S copies in the preimage of the embedding. In the sequel, we will avoid some notational unpleasantness by agreeing that $\lambda_1, \lambda_2, \ldots$ always denote bookkeeping vertices and that $u_1^*, u_2^*, \ldots u_k^*$, $k \leq S$, always denote copies u.

It is known that for every S, T > 0, there is an ancestor tree which cannot be simulated with space and time dilation S and T by any 1-exit program and for every S, T there is a 1-exit program which cannot be simulated with space and time dilation S and T by any 1-label program. This is very suggestive of a hierarchy in the number of labels for the $\leq_{S,T}$ relation, among ancestor trees.

We can now show that such hierarchies collapse. That is, we show that for every k > 1, there is a T $\geq$ 0 such that every k-label program can be simulated by some 1-label program with space dilation S = 1 and time dilation T = T(k). We begin by considering a general embedding procedure which dilates space by S(k) > 1, since this result is technically easier.

AN OBSERVATION

Let G be a binary tree with root x and consider a vertex y which is not the father of two vertices. G can be modified by viewing y as the root and inverting the father-son relationship along the path from x to y. Obviously, the resulting graph is still a binary tree; we denote this tree by G^y.

MAIN SIMULATION RESULT

Let H be an ancestor tree and choose a vertex x of H such that among the descendents of x there is exactly one label y. Let H' be obtained from H by replacing the subtree H_x by the graph shown in Figure 1.

In this graph, K is H_x with its vertices subscripted by "1", L is $(H_x - H_y)^{f_H(y)}$ with its vertices subscripted by "2", and M is H_y with its vertices subscripted by "2". Thus α is a "second copy" of $f_H(y)$. In addition,

each arc $u \xrightarrow[a]{} y$ or $u \xrightarrow[a]{} x$ is replaced by $u_1 \xrightarrow[a]{} \lambda_2$, $u_2 \xrightarrow[a]{} \lambda_2$, while every $u \xrightarrow[a]{} v$ with $v \neq x,y$ is replaced by $u_1 \xrightarrow[a]{} v$ and $u_2 \xrightarrow[a]{} v$. Then we have $H \leq_{2,3} H'$, which may be proved easily by a case analysis of the possible arcs in H and their copies in H'. Note further that if x is not a label, then H' has the same number of labels as H, while if x is a label, the total number of labels is decreased by one.

We now prove that any k-label program ($k \geq 2$) can be simulated by a (k-1)-label program. To this end, let H be a k-label program, $k \geq 2$. Two cases arise.

Case I. Some vertex x contains exactly one label in each of its subtrees. If the root of either subtree is a label, no transformation is required for that subtree. In all other cases, replace each subtree as above to yield a k-label program H', where $H \leq_{2,3} H'$ and in H' the sons of x are both labels. Let the sons of x by $y,z \in V(H')$. Now, clearly each arc $u \xrightarrow[a]{} y$ or $u \xrightarrow[a]{} z$ can be replaced by an arc $u \xrightarrow[a]{} x$ at the expense of dilating path lengths by one arc. Hence, this transformation has the effect of replacing the pair of labels y,z by a single label x. If H'' is the result of such a transformation, then since $d_{H''}(x,y) = d_{H''}(x,z) = 1$, we have $H \leq_{2,3} H''$.

Case II. Some label vertex x has exactly one label y as a descendent. The transformation given above when applied to the subtree rooted at x yields a k-1 label program H' such that $H \leq_{2,3} H'$.

Thus, every k-label program is simulated with S = 2, T = 4 by a k-1 label program. We have immediately that every k-label program H is simulated by some 1-label program G with S, T independent of $|V(H)|$; more specifically

$S = 2^{k-1}$, $T = 4^{k-1}$. It is easily seen that for every S, T there is a 1-label program which cannot be simulated by any 0-label program (i.e., by a binary tree).

AN IMPROVEMENT

The vertex duplication in the construction above is somewhat artificial; it is used only to keep track of "end points" of circuits, and we might try to use some inherent symmetry in the problem to avoid such duplication. In fact, such duplication need never be introduced. That is, we can prove that for every k-label program H, ($k \geq 2$) there is a (k-1)-label program H' such that $H \leq_{1,4} H'$.

Let B_n denote the regular graph on n vertices with degree 2, shown in Figure 2. If $V(B_n) = \{a_1, \ldots, a_n\}$, then $B_n \leq_{1,3} G_0$; and $B_n \leq_{1,4} G_1$, when G_0 and G_1 are as shown in Figures 3(a) and 3(b), respectively.

The first simulation is apparently the better of the two, but in fact the simulation $H \leq_{1,4} G_1$ is the one which is to be preferred for the simulation to be described.

Using this transformation, given a tree H, as shown in Figure 4(a), we embed $H \leq_{1,4} H^*$ where H^* is as in Figure 4(b). As in the case $B_n \leq_{1,4} G$, we now have a_1 and a_n relatively close to each other. For obvious reasons, we call this transformation a *folding* of H.

Now suppose that H is a subtree of a k-label program H_o ($k \geq 2$) that a_1 and a_n are both labels, $V(H_1)$ has no other label, and none of $a_2, \ldots, a_{n-1}$ is a label. We then form H' by folding H and replacing each $u \xrightarrow{a} a_n$ by $u \xrightarrow{a} a_1$. If no such subtree H of H_0 exists, then no label is related to any other as either

a descendent or an ancestor. But since each $\{x,y\} \subseteq V(H_0)$ share a common ancestor (viz. the root of H_0) choose any two labels x,y and let z be their deepest common ancestor. This identifies subtrees of the form H with $a_1 = z$ and $a_n \in \{x,y\}$. Fold each of these subtrees and replace each arc $u \xrightarrow[a]{} x$ or $u \xrightarrow[a]{} y$ by $u \xrightarrow[a]{} z$. Then, if the resulting ancestor tree is H', we have $H_0 \leq_{1,4} H'$.

Note that the passage from a k-label program to a 1-label program still requires $T = 4^{k-1}$. It is not known if this is (asymptotically) the best possible.

REFERENCES

1. R. Lipton, S. Eisenstat, R. DeMillo, "Space and Time Hierarchies for Classes of Control Structures and Data Structures", *Journal of the ACM*, Vol. 23, No. 4, Oct 1976, pp. 720-732.

2. R. DeMillo, S. Eisenstat, R. Lipton, "Space-Time Tradeoffs in Structured Programming: An Improved Combinatorial Embedding Theorem" (to appear); parts of this paper appear as "Space-Time Tradeoffs in Structured Programming", *Proceedings of the 1976 Johns Hopkins Conference in Information Sciences and Systems*, pp. 431-434.

3. S. R. Kosaraju, "Analysis of Structure Programs", *Journal of Computer and System Sciences*, Vol. 9, No. 3, Dec 1974, pp. 232-255.

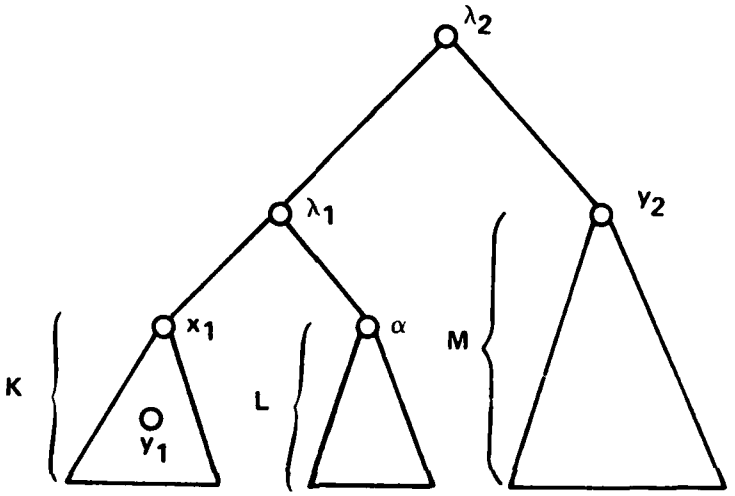


Figure 1. Modification of H

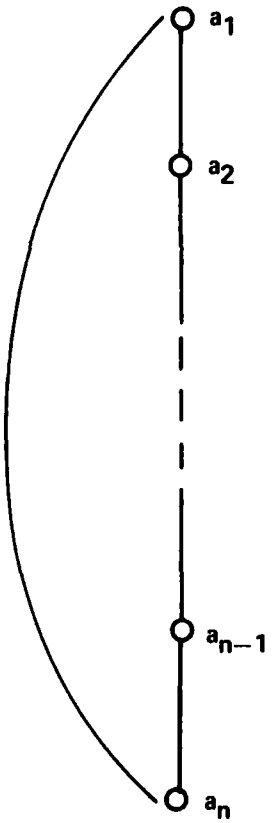


Figure 2. The Graph B_n

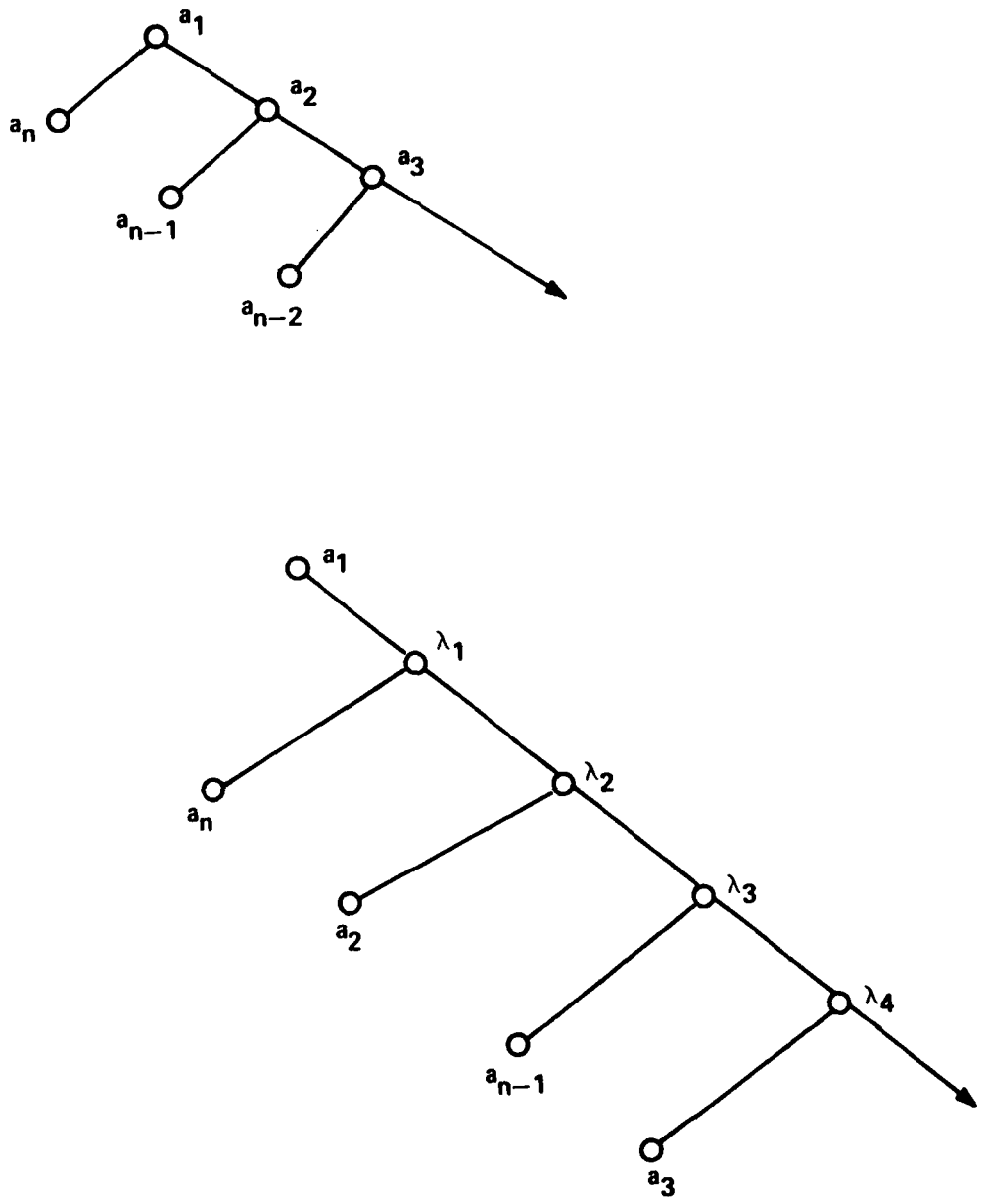


Figure 3. The Graphs
G_0 (upper) and
G_1 (lower)

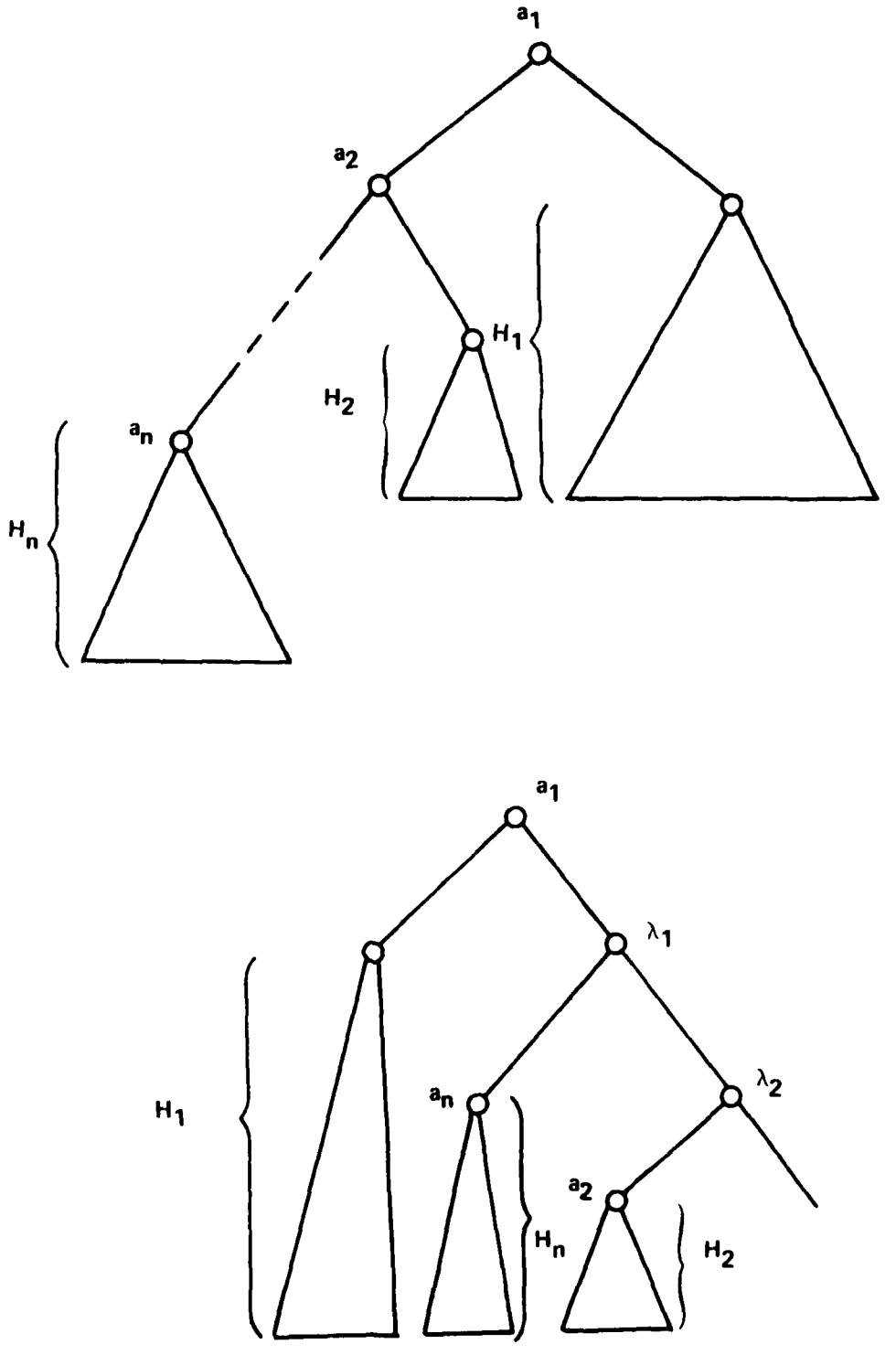


Figure 4. The trees H (upper) and H* (lower)

Programming Techniques
S. L. Graham, R. L. Rivest, Editors

Preserving Average Proximity in Arrays

Richard A. DeMillo
Georgia Institute of Technology
Stanley C. Eisenstat and Richard J. Lipton
Yale University

Programmers and data structure designers are often forced to choose between alternative structures. In storing these structures, preserving logical adjacencies or "proximity" is usually an important consideration. The combinatorial problem of storing arrays as various kinds of list structures is examined. Embeddings of graphs are used to model the loss of proximity involved in such storage schemes, and an elementary proof that arrays cannot be stored as linear lists with bounded loss of proximity is presented. Average loss of proximity is then considered, and it is shown that arrays cannot be stored as linear lists with only bounded loss of average proximity, but can be so stored in binary trees. The former result implies, for instance, that row major order is an asymptotically optimal storage strategy for arrays.

Key Words and Phrases: arrays, graph embedding, linear lists, proximity, average proximity, trees
CR Categories: 4.34, 5.24, 5.25, 5.32

This work was supported in part by the US Army Research Office under grants DAHC04-74-G-0179 and DAAG29-76-G-0338, by the Office of Naval Research under grant N00014-67-0097-0016, and by the National Science Foundation under grant DCR74-12870 Authors' current addresses: R. DeMillo, School of Information and Computer Science, Georgia Institute of Technology, Atlanta, GA 30332; S. Eisenstat and R. Lipton, Computer Science Department, Yale University, New Haven, CT 06520

1. Introduction

Efficient algorithms often require specific data structures on which to operate. In many of the algorithms of [1], for instance, the running time depends critically on the number of probes of the data structure needed to access a single data item. In practical computation, however, many factors interfere with optimal data organization. Among the most important of these factors is the locality of references in data structures. Since most data structures are arranged linearly in memory, a set of local references to the structures can require accessing of data elements widely separated in memory. This is particularly relevant in paging environments where a page fault may occur during access of elements which are logically adjacent in a data structure.

In [6], Rosenberg examined the problem of storing arrays as a linear structure with bounded loss of proximity between data elements and showed that any such storage scheme must, in the worst case, induce unbounded loss of proximity. In previous papers [2, 4], we considered the proximity-preserving issue in a more general setting. We used graphs to represent data structures as described in [3]: vertices represent data elements or nodes, and edges represent logical adjacencies; if G and H are graphs, we write $G \leq_T H$ if G can be "stored" as H so that no adjacent nodes of G are more than a distance T apart in H. Using this model we were able to show that if G_n is an $n \times n$ array and H is any of a very general type of structure (including as subcases linear lists, binary trees, and threaded lists [5]), then $G_n \leq_{T(n)} H$ only if

$$T(n) \geq (1/3) \log n - 2/3. \quad (1)$$

In particular, (1) shows that arrays cannot be stored in trees or lists with bounded loss of proximity, extending the result of Rosenberg [6].

The lower bound represented by inequality (1) is a worst-case result. Since algorithms which manipulate data structures tend to "look at" all of the data, it seems natural to investigate the *average* loss of proximity involved in storing arrays as various list structures. We find a distinction between linear memory and arbitrary lists: arrays *can* be stored as nonlinear lists with bounded loss of average proximity, but cannot be so stored in linear memory.

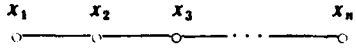
2. Graphs and Embeddings

By a graph $G = (V, E)$ we mean a set V of *vertices* and a set E of unordered pairs of vertices, the *edges*. Note that all graphs we discuss are undirected, i.e. $\{x, y\} \in E$ iff $\{y, x\} \in E$. An edge between two vertices x and y is represented

A *path of length n* between $x, y \in V$ exists if there is a sequence of edges $\{x_0, x_1\}, \{x_1, x_2\}, \ldots, \{x_{n-1}, x_n\}$ with $x_0 = x$ and $x_n = y$. If $G = (V, E)$ and $x, y \in V$, then $d_G(x, y)$ is defined to be the length of a minimal length path between x and y. A graph G is intended to model a data structure; therefore we think of the vertices of G as *nodes* of the structure and edges of G as representing logical adjacencies in the structure.

We shall consider three classes of data structures, defined by the classes of graphs which represent them.

Lines. A line L is a graph $(\{x_i\}_{i \in \{1, \ldots, n\}}, E)$ with $\{x_i, x_j\} \in E$ iff $j = i + 1$. For example,



represents a line with n nodes.

Binary Trees. A binary tree is a graph which is either empty or consists recursively of a root connected by edges to the roots of binary trees called the left and right subtrees of the root. We assume the elementary properties of trees as described in [1, 3].

Arrays. An $n \times n$ array G_n is a graph $(\{x_{i,j}\}_{i,j \in \{1, \ldots, n\}}, E)$ with $\{x_{i,j}, x_{i,j+1}\} \in E$ and $\{x_{j,i}, x_{j+1,i}\} \in E$ for all $1 \leq i \leq n$ and $1 \leq j < n$. The 3×3 array G_3 is shown in Figure 1. Modeling $n \times n$ arrays as graphs of the form G_n does not recover the notion of "randomly" accessing array elements. But, as noted in [4, 6], algorithms which operate on arrays tend to access array elements "locally," e.g. along rows and columns or, as in the Strassen matrix multiplication algorithm, as 2×2 subarrays [1]. For these purposes the above representation of arrays is entirely faithful to the adjacency properties of arrays.

Given two classes of graphs X and Y which represent distinct data structures, it is often natural to ask whether or not each $G \in X$ can be "represented" as some $H \in Y$ in such a way as to preserve the accessing characteristics of G. Furthermore, we may want such a representation of G to *preserve proximity*; that is, the representation of G in H should be such that adjacent nodes in G are represented within some bounded distance of each other in H. The need for preserving proximity in this fashion has been defended by Rosenberg [6].

The combinatorial aspects of this sort of representation are recovered by the following definition (see Figure 2):

Definition. Let $G = (V, E)$ and $G^* = (V^*, E^*)$ be graphs, and let $T \geq 1$. We say that G is *T-embeddable* in G^* (written $G \leq_T G^*$) if there is an injection $\Phi: V \to V^*$ (called an *embedding*) such that for all $\{x, y\} \in E$, $d_{G^*}(\Phi(x), \Phi(y)) \leq T$.

Example. Let $K_n = (\{x_1, \ldots, x_n\}, E)$ be the complete graph on n nodes; i.e. $\{x_i, x_j\} \in E$ for all $1 \leq i, j \leq n$ with $i \neq j$. Let H be the graph shown in Figure 3. Then it is easily verified that $K_n \leq_2 H$ via the embedding $\Phi(x_i) = y_i$ for $i = 1, \ldots, n$.

Fig. 1. Graph G_3.

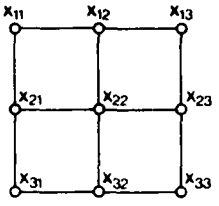
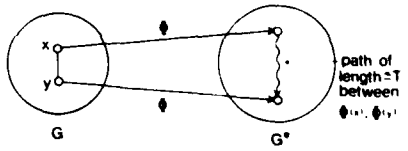


Fig. 2. Illustration of $G \leq_T G^*$.



A main result of [4] is that arrays cannot be stored as binary trees with only bounded loss of worst-case proximity. The precise statement of this result is

THEOREM 1. *Let $\{G_n\}_{n \geq 1}$ be the class of $n \times n$ arrays. If $G_n \leq_{T(n)} H$ for some binary tree H, then*

$$T(n) \geq \log n - 3/2.$$

For completeness, we sketch the proof of Theorem 1 (cf. [4]). We require an additional definition, which will also be useful in a later result.

Definition. Let $G = (V, E)$ be a graph and let $A \subseteq V$. The *boundary* $\partial(A)$ of A is the set of vertices in A adjacent to vertices in $V - A$, i.e.

$$\partial(A) = \{x \in A: \text{for some } y \in V - A, \{x, y\} \in E\}.$$

A key property of the boundary operator is given by the following lemma (see [4]).

BOUNDARY LEMMA. *If $G_n = (V_n, E_n)$ is an array and $A \subseteq V_n$ is such that $|A| \leq n^2/2$,[1] then*

$$|A| \leq 2\,|\partial(A)|^2.$$

PROOF OF THEOREM 1. Assume $G_n \leq_{T(n)} H$ where $G_n = (V_n, E_n)$, and let $H = (V, E)$ be as shown in Figure 4, where $H_1, \ldots, H_k$ are subtrees of H and $H_k = (V^{(k)}, E^{(k)})$ has the following property:

$$n^2/4 \leq |\{x \in V_n: \Phi(x) \in V^{(k)}\}| \leq n^2/2.$$

Let A_k denote $\{x \in V_n: \Phi(x) \in V^{(k)}\}$. Then $|A_k|$ satisfies the hypothesis of the Boundary Lemma, and thus

$$|\partial(A_k)| \geq (1/\sqrt{2})|A_k|^{1/2} \geq n/2\sqrt{2}.$$

By definition, if $x \in \partial(A_k)$, then $d_{G_n}(x, y) = 1$ for some $y \in V - A_k$, and thus, since $G_n \leq_{T(n)} H$,

$$d_H(\Phi(x), \Phi(y)) \leq T(n).$$

But

$$|\{v \in V^{(k)}: d_H(u, v) \leq T(n)$$

[1] If S is a set, $|S|$ denotes the cardinality of S.

Fig. 3. Graph H.

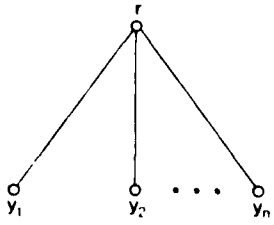
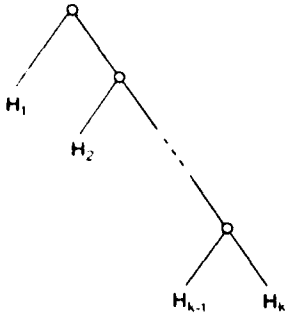


Fig. 4. Decomposition of H.



for some $u \in V - V^{(k)}| \leq 2^{T(n)}$;

therefore

$$2^{T(n)} \geq |\partial(A_k)| \geq n/2\sqrt{2},$$

or

$$T(n) \geq \log n - 3/2. \qquad \square$$

Actually, in [2] we prove a much stronger result: If we allow embeddings of $n \times n$ arrays which not only preserve proximity by a factor of $T(n)$ but which also may "split" a vertex at most $S(n)$ times, then

$$T(n) + \log \log S(n) \geq c \cdot \log n$$

where c is a fixed positive constant independent of n.

3. Preserving Average Proximity

The relation $\leq_T$ represents a worst-case analysis of proximity-preserving transformations. Since data structures are frequently accessed "uniformly" in that the probability of access to a particular node by a particular edge is uniformly distributed, Theorem 1 leaves open the question of whether or not there are ways to store arrays as various other structures which, *on the average*, preserve proximity. To investigate this problem we need an additional definition.

Definition. Let $G = (V, E)$ and $G^* = (V^*, E^*)$ be graphs. We say that G is *A-average embeddable* (written $G \leq_A^{\text{ave}} G^*$) if there is an embedding $\Phi: V \to V^*$ such that

$$\sum_{(x,y)\in E} d_{G^*}(\Phi(x), \Phi(y)) \leq A \cdot |E|.$$

We first consider the case of storing an array as a linear structure, such as a linear list or in linear memory.

THEOREM 2. *Let $\{G_n\}_{n\geq 1}$ be the class of $n \times n$ arrays. If $G_n \leq_{A(n)}^{\text{ave}} L$ for a line L, then*

$$A(n) \geq n/12.$$

PROOF. Assume $G \leq_{A(n)}^{\text{ave}} L$ where $L = (\{x_1, \ldots, x_m\}, E)$ is a line. It is clearly sufficient to assume that $m = n^2$ since, if $G_n \leq_{A(n)}^{\text{ave}} L$ by an embedding Φ and $x_k \notin \Phi(V_n)$, then Φ also defines an $A(n)$ average embedding of G_n into the line with x_k removed.

Let $D_i = \{\Phi^{-1}(x_j): 1 \leq j \leq i\}$ for $1 \leq i \leq n^2/2$ so that $|D_i| = i \leq n^2/2$ (see Figure 5). For each $y \in \partial(D_i)$ there is an edge between y and some node not in D_i; hence there is a path between $\Phi(y)$ and some node not in $\Phi(D_i)$ which must pass through x_i. Since $|D_i| \leq n^2/2$, by the Boundary Lemma each $\Phi(D_i)$ makes a contribution of at least $(|D_i|/2)^{1/2}$ to

$$\sum_{(x,y)\in E_n} d_G(\Phi(x), \Phi(y)).$$

For suppose $\Phi^{-1}(x_k)$ and $\Phi^{-1}(x_l)$ $(k < l)$ are adjacent in G_n. Then the $l - k + 1$ that this adjacency should add to the sum accrues by x_k's membership in $\Phi(D_k)$, $\Phi(D_{k+1}), \ldots, \Phi(D_{l-1})$, each membership contributing 1 to the sum.

Therefore

$$\sum_{(x,y)\in E_n} d_G(\Phi(x), \Phi(y)) \geq \frac{1}{\sqrt{2}} \sum_{i=1}^{n^2/2} |D_i|^{1/2}$$

$$= \frac{1}{\sqrt{2}} \sum_{i=1}^{n^2/2} \sqrt{i} \geq \frac{1}{\sqrt{2}} \int_0^{n^2/2} (\sqrt{x})dx = \frac{n^3}{6}.$$

Hence

$$n^3/6 \leq A(n)|E_n| = A(n)(2n^2 - 2n),$$

which yields $A(n) \geq n/12$. $\square$

An interesting interpretation of this result is that any reasonable sequential method of array storage is *asymptotically* optimal with respect to proximity. Consider, for instance, an embedding Φ of G_n into a line L which places nodes of G_n in row-major order; that is, $\Phi(x_{i,j}) = x_{(i-1)+(j-1)n}$. With this embedding into L,

$$G_n \leq_{(n+1)/2}^{\text{ave}} L.$$

By Theorem 4.2 of [6], row-major storage is also optimal for worst-case proximity.

In contrast to Theorems 1 and 2, we have the following.

THEOREM 3. *When n is a power of 2, there is a binary tree H such that $G_n \leq_8^{\text{ave}} H$.*

PROOF. Let $G_n = (V_n, E_n)$ be given and suppose $n = 2^k$ for some k. We shall describe a recursive method of embedding G_n into a complete binary tree. Divide G_n into four $n/2 \times n/2$ subarrays (see Figure 6(a)) and attach the subarrays as leaves of a complete binary tree H as shown in Figure 6(b)).

Fig. 5. Embedded edges in L.

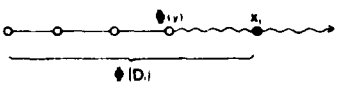
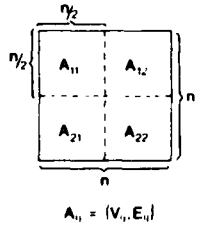
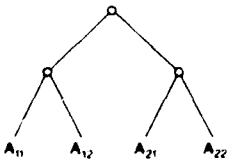


Fig. 6. Illustration of embedding of Theorem 3. (a) Recursive decomposition of G_n. (b) Embedding G_n into complete binary tree.





Clearly

$$\sum_{(x,y)\in E_n} d_H(\Phi(x),\Phi(y)) \le N + \sum_{A_{ij}} \sum_{(x,y)\in E_{ij}} d_H(\Phi(x),\Phi(y)), \quad (2)$$

where N is the sum of the lengths of the $2n$ paths between the $4n - 4$ nodes along the boundaries of the A_{ij}. By continuing this process recursively, we can suppose that the A_{ij} of Figure 6(b) are themselves complete binary trees, and therefore we may bound N in inequality (2) by

$$N \le 2n[2 \log (n^2/4) + 4] = 8n \log n.$$

This leads to the following recurrence for $f(n)$, the sum of the lengths of paths which correspond under Φ to embedded edges of an $n \times n$ array:

$$f(1) = 0$$

$$f(n) \le 4f(n/2) + 8n \log n.$$

The solution to this recurrence satisfies

$$f(n) \le 16(4^{\log n}) - 8n \log n - 16n.$$

Thus

$$A = \sum_{(x,y)\in E_n} d_H(\Phi(x),\Phi(y)) \Big/ |E_n| \le \frac{f(n)}{2(n^2 - n)} \le \frac{8n^2 - 4n \log n - 8n}{n^2 - n} \le 8. \quad \square$$

For n not a power of 2, Theorem 3 clearly can be modified to hold with a slightly larger constant.

Even though the number of vertices reachable by a path of length k grows as 2^k in a complete binary tree versus k^2 in an array, it is important to note that not *any* embedding technique will work in the proof of Theorem 3. Similarly, not every graph in which fewer than 2^k vertices can be reached by paths of length k can be embedded by using the recursive decomposition of Theorem 3; the graph must have the property that it can be "cut" into regions with boundaries that are not too large. A basic question to be resolved is whether or not any family of graphs with neighborhoods growing slower than 2^k are A-average embeddable in trees for some constant A.

Acknowledgments. We would like to thank Arnold Rosenberg for his thoughtful comments on a draft of this paper and Larry Snyder for several helpful suggestions.

Received March 1976; revised January 1977

References

1. Aho, A., Hopcroft, J., and Ullman, J.D. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, Reading, Mass., 1974.
2. DeMillo, R.A., Eisenstat, S.C., and Lipton, R.J. Space-time tradeoffs in structured programming. Proc. 1976 Conf. on Inform. Sci. and Syst., Baltimore, Md., 1976, pp. 431-434.
3. Knuth, D.E. *The Art of Computer Programming, Vol. 1: Fundamental Algorithms*. Addison-Wesley, 1968, p. 305ff.
4. Lipton, R.J., Eisenstat, S.C., and DeMillo, R.A. Space and time hierarchies for classes of control structures and data structures. *J. ACM 23*, 4 (1976), 720-732.
5. Perlis, A.J., and Thornton, C. Symbol manipulation by threaded lists. *Comm. ACM 3*, 4 (1960), 201-202.
6. Rosenberg, A.L. Preserving proximity in arrays. *SIAM J. Comptng. 4*, 4 (1975), 443-460.
7. Rosenberg, A.L. Managing storage for extendible arrays. Proc. Sixth ACM Symp. on the Theory of Comptng., Seattle, Wash., 1974, pp. 293-302.

Corrigendum. Turing Award Lectures

Michael O. Rabin, "Complexity of Computations," *Comm. ACM 20*,9 (September 1977), 625–633.

Page 626, first paragraph, second sentence: replace ":" after "communications" by ";". In column 1, line 16 should begin "...Considerable...". In Column 2, the first term of equation (2) should be a_{ij} and the first word that follows that equation is "of". On page 627, the title to section 2.5 begins with "Sorting."

On page 628, in column 2, line 21, $\mu(s)$ should be $\mu(S)$. In column 2 of page 629, line 15 has a comma before "holds" and on line 18 "E. Blum" should be "M. Blum."

On page 630, column 1, the expression on the first and second lines should read "$\alpha_0(a_0,\ldots,a_n),\ldots,\alpha_n(a_0,\ldots,a_n)$", and on line 8 the author is Motzkin. On page 631, column 1, line 9, "O(n^e)" should be "o(n^e)". In column 2, line 12 should read "such that for every $n_0 < n$" and line 14 should read "satisfying (1) ℓ (H) = n, and (2)...". The theorem's statement should end with "Here ℓ (H) denotes the length of H."

Line 4 of column 1 of page 633 should read "secrecy.", and the following paragraph should end with a question mark.

THE AVERAGE LENGTH OF PATHS EMBEDDED IN TREES*

Richard A. DeMillo
School of Information and Computer Science
Georgia Institute of Technology
Atlanta, Ga. 30332

Richard J. Lipton
Department of Computer Science
Yale University
New Haven, Ct. 06520

A graph, G, consists of vertices V(G) and edges E(G); paths are sequences of vertices connected by edges, and path length is defined by the number of edges along the path. For $x,y \in V(G)$ we use $d_G(x,y)$ to denote the length of a minimal length path between x and y, if such a path exists. An n × n array, G_n, consists of vertices $V(G_n) = \{x_{ij}\}_{i,j \leq n}$ and edges which, except at the obvious extremal conditions, are linked as follows:

* The work of both authors was supported in part by the U.S. Army Research Office, Grant No. DAAG29-76-G-0338.

$$(x_{i,j}, x_{i+1,j}) \in E(G_n), \text{ and}$$

$$(x_{i,j}, x_{i,j+1}) \in E(G_n).$$

Such graphs are also called <u>rook-connected</u>. A binary tree is as defined in [1,2]; that is, a binary tree H is a connected acyclic graph with a designated <u>root</u> and ancestor - descendent relation defined so that each $x \in V(H)$ has at most two immediate descendents.

Let us write $G \leq_T H$ when there is a one-one mapping (called an <u>embedding</u> of G into H $\Phi : V(G) \to V(H)$, such that for all $(x,y) \in E(G)$,

$$d_H(\Phi(x), \Phi(y)) \leq T.$$

As described in [1], it follows from simple volumetric arguments that for all $T > 0$, there exists a binary tree H such that $H \not\leq_T G_n$, for all $n \geq 1$. The corresponding intuition for $G_n \leq_T H$ does not hold. It would now seem that since in G_n

$$| \{x \in V(G_n): d_{G_n}(x,y) \leq k\} | = O(k^2) \qquad (1)$$

while in a complete binary tree H

$$| \{x \in V(H): d_H(x,y)\} | \geq 2^{k-1} \qquad (2)$$

that $G_n \leq_T H$ would now be possible for some bounded T. It is therefore somewhat surprising that $G_n \leq_T H$ only if

$$T \geq \log n - 1.5$$

(See [1], for details).

It is still obvious from inspection that neighborhoods in trees can be much more densely growing than neighborhoods in arrays, and therefore by choosing a suitably global measure of loss of proximity, this difference should be distinguishable. In [2] we considered such a measure:

$G \leq_{A}^{\text{edge}} G^*$ if for some embedding $\Phi : V(G) \to V(G^*)$

$$\sum_{(x,y) \in E(G)} d_{G^*}(\Phi(x), \Phi(y)) \leq A \mid E(G) \mid .$$

It follows [2] that for $b = 8.5$

$$G_n \leq_{b}^{\text{edge}} H$$

for some binary tree H. This upper bound can be improved to $b = 7 - o(1)$†

The relation $\leq_{A}^{\text{edge}}$ may be thought of as <u>averaging</u> - with relative frequencies uniformly distributed to the edges E(G) - over the <u>edges</u> of G. We now make a more global definition which finally may be used to recover our original, although imprecise, intuitions about path lengths in binary trees. We will essentially average over shortest paths:

$G \leq_{A}^{\text{paths}} G^*$ if one is an embedding $\Phi : V(G) \to V(G^*)$ such that

$$\Gamma_n \leq A \cdot \Delta_n$$

where

$$\Gamma_n = \sum_{\Phi(x),\Phi(y)} d_{G^*}(\Phi(A,\Phi(y))$$

† L. Snyder, private communication.

and

$$\Delta_n = \sum_{x,y} d_G(x,y).$$

We then have the following theorem.

Theorem. For each $n \geq 0$, let A_n be the least real number such that

$$G_n \leq \overset{\text{paths}}{A_n} H,$$

for a binary tree H. Then

$$\lim_{n \to \infty} A_n = \lim_{n \to \infty} \Gamma_n / \Delta_n = 0.$$

Proof we first show

$$\Delta_n = \Omega(n^5)$$

Let us choose $B_1, B_2 \subseteq V(G_n)$ so that

$$B_1 = \{x_{ij} : 1 \leq i, j \leq n / 4\}$$

$$B_2 = \{x_{ij} : \frac{3n}{4} \leq i,j \leq n\}$$

so that $|B_1 \times B_2| = [n^2 / 16]^2$. Now clearly, for any $(x,y) \in B_1 \times B_2$

$$d_{G_n}(x,y) \geq n / 2,$$

and hence by definition

$$\Delta_n \geq n^5 / 512 = \Omega(n^5)$$

We now obtain the following upper bound for Γ_n

$$\Gamma_n = O(n^4 \log n).$$

As in [2] let $A_{ij} \subseteq V(G_n)$, $1 \leq i, j \leq 2$, $|A_{ij}| = n^2 / 4$,

Denote the $n / 2 \times n / 2$ decomposition of G_n and notice that

$$\Gamma(n) \leq 4 \Gamma\left(\frac{n}{2}\right) + \frac{1}{2} n^4 \log n.$$

Thus $\Gamma(n) \leq \alpha n^4 \log n + \beta n^4$ from which the theorem follows directly.

1. R.J. Lipton, S. Eisenstat, R.A. DeMillo, "Space and Time Hierarchies for Classes of Control Structures and Data Structures", Journal of the ACM, Vol. 23, No. 4, Oct. 1976, pp. 720-732.

2. R.A. DeMillo, S.C. Eisenstat, R.J. Lipton, "Preserving Average Proximity in Arrays", Communications of the ACM (to appear).

ON SMALL UNIVERSAL DATA STRUCTURES AND RELATED COMBINATORIAL PROBLEMS †

(Preliminary Report)

Richard DeMillo

School of Information and Computer Science
Georgia Institute of Technology
Atlanta, GA 30332

Stanley Eisenstat
Richard J. Lipton

Department of Computer Science
Yale University
New Haven, CT 06520

† Work supported in part by U.S. Army Research Office, Grant No. DAAG29-76-G-0338, and by the Mathematics Research Center, University of Wisconsin-Madison.

INTRODUCTION

One of the most significant changes in theoretical computer science has been the recent infusion of the methods and problems from combinatorial analysis. Among the most powerful combinatorial theorems which have been imported to computer science are those of extremal graph theory [1]: in extremal graph theory, one is interested in the *largest* (or in complementary problems, the *smallest*) graph which avoids (or contains) a given structure. Purely combinatorial results (which have significance, e.g., for the design of circuit boards) have been obtained by Chung and Graham [2] and by Chung, Graham, and Pippenger [3]. In this paper, we extend this theory to encompass results concerning data structures.

As motivation for the results to the described, note that many of the large data structures manipulated by the programs described in [4,5] have two characteristics

(i) they are sequentially accessed, and

(ii) many distinct structures convolve in the same physical memory.

For applications of this sort, it would obviously be desirable to have available a *universal data structure* in which all data structures from a given class may gracefully reside. In view of (i), by "graceful" we mean that the sequential accessing characteristics of the embedded data structures are not too drastically altered. Let us *measure* such alterations by the *dilation* of logical adjacencies [6,7] needed to embed all structures from a given class into a universal structure; this is then a complementary extremal graph theory problem: what is the size (number of edges) of the smallest universal graph for a given dilation factor.

The main results contained in this paper address such problems from a number of points of view.

(1) We give several asymptotically optimal universal data structures for graphs of n vertices when average dilation [7] is used as a measure.

(2) We discuss a universal data structure for graphs of n vertices where worst-case dilation is used as a measure [6].

(3) We consider variations of the *average dilation measure* which gives favorable comparisons between data structures studied in [6,7].

(4) We consider the kinds of "sharing" that can take place between "almost linear" and "almost complete tree-like" structures.

(5) Finally, we propose a data structure embedding model which recovers some aspects of random accessing of data items, and prove a space-time tradeoff which seems to indicate that no savings is possible in RAM models which assess accessings costs uniformly [8].

PRELIMINARIES

A graph, G, is defined by its *vertices*, V(G), and *edges*, $E(G) \subseteq V(G) \times V(G)$. Edges are assumed to be *undirected*: a pair of vertices x,y are *connected* if either $(x,y) \in E(G)$ or $(y,x) \in E(G)$. A *path* between x_0, x_n is said to be of *length n*. The distance *metric* $d_G(x_0, x_n)$ is defined to be n if there is no shorter path than $x_0,\ldots,x_n$.

A graph represents a *data structure* in the obvious way: vertices represent nodes or records and connectedness models logical adjacency. The following relations and their significance for data structures can be found in [6,7]. Let G, G* be graphs. We say that G is T-*worst case* embeddable in G* ($G \leq_T G^*$) if there is a one-one $\Phi: V(G) \to V(G^*)$ such that $(x,y) \in E(G)$ implies

$$d_{G^*}(\Phi(x), \Phi(y)) \leq T. \qquad (1)$$

Similarly, G is A-*average case* embeddable in G* ($G \leq_A^{avg} G^*$) if there is a one-one Φ as above such that

$$\sum_{\substack{x,y \\ \text{connected}}} d_{G^*}(\Phi(x), \Phi(y)) \leq A \cdot |E(G)|. \qquad (2)$$

In [4,5], comparisons between several natural classes of graphs give asymptotic bounds on T, A in (1), (2) as functions of $|V(G)|$. Shortly after the announcement of the results of [6], R. M. Karp suggested to us the following class of problems connected with extremal graph theory: what are the characteristics of $\leq_T$ - universal data structures; i.e., those structures which T-worst case embed all graphs in a given class. This paper grew out of considering these problems.

UNIVERSAL GRAPHS

Let ζ^n be a given class of graphs G, $|V(G)| = n$. Let us ask about a data structure which is $\leq_T$ or $\leq_A^{avg}$ *universal* for ζ^n. In particular, let us define

$$w(\zeta^n, T) = \underline{\min}\, \{|E(G)|: \ G^n \in \zeta^n, G^n \leq_T G\} \qquad (3)$$

and

$$a(\zeta^n, A) = \underline{\min}\, \{|E(G)|: G^n \in \zeta^n, G^n \leq_A^{avg} G\}.$$

For T = 1, (3) becomes the complementary extremal graph problem studied in [2,3].

By an *n-tree* G, we mean a connected acyclic graph G, with $|V(G)| = n$. It is also convenient to think of trees as *rooted* in the following sense: accompanying G, there is an ancestor-descendent relation that assigns direct ancestors and direct descendents to vertices in the obvious way so that a vertex with no ancestors can be designated as the *root* of the tree.

(Obviously this choice is not going to be unique, but we assume that G is not characterized until such a choice is made). A d-ary n-tree is an n-tree in which each vertex has *at most* d direct descendents. We denote, respectively, the classes of n-trees and d-ary n-trees by Γ^n and Γ_d^{n*} .

By [2] it is known that $\frac{1}{2}n \log n < w(\Gamma^n, 1) < n^{1+k(n)}$, $k(n) = [\log \log n]^{-1}$. †

The upper bound was improved in [3] to

$$w(\Gamma^n, 1) = O(n \log n[\log \log n]^2)$$

The bounds on $a(\Gamma^n, 1)$ are apparently not elsewhere considered.

Superficially, at least, all interest in further characterization of (3) is destroyed by the following obvious

Theorem. For $T \geq 2$

$$w(\Gamma^n, T) = n$$

Of course, in (3), the "target" graph G may have unbounded degree. Therefore, it is natural to consider $w(\zeta^n, T, S)$ and $a(\zeta^n, T, S)$ where in both cases the target graph G is restricted to be in the set S. Note that now the theorem just cited is no longer obviously true.

---

* Thus Γ_2^n = binary trees on n vertices.

† In the sequel, we use log x for $\log_2 x$ and ℓnx for $\log_e x$.

The best that is known is the upper bound of [3] (S = all cubic graphs)

$$w(\Gamma^n, 1, S) \leq \frac{2\sqrt{2}}{n} \exp(\log^2 n/2 \log 2) . \qquad (4)$$

It is not obvious that when (i) "targets" are restricted to binary trees and (ii) $w(\Gamma_2^n, T, \Gamma_2^n)$ is considered, that it is possible to do any better than the union of all trees in Γ_2^n, giving a structure of size $4^n/2\pi n$. But, we have the following

Theorem. For each $T \geq 1$, there is a binary tree H, such that $G \geq_T H$ for all $G \in \Gamma_2^n$, and

$$\ell n|E(G)| \leq \frac{\ell n^2 n}{\ell n 4} ;$$

or in other words

$$w(\Gamma_2^n, T, \Gamma_2^n) = \exp \frac{1}{\ell n 4} (\ell n\ n)^2 + O((\ell n\ n)^2) .$$

A key step in the proof of this theorem hinges on the solution to the fascinating "almost linear" recurrence

$$u_n = u_{n-1} + u_{\lceil \frac{n}{2} \rceil} , \qquad (5)$$

first considered by Knuth [9]. This also establishes a connection between the theorem and ineq. (4): u_n is also the number of partitions of 2n of the form $\Sigma \alpha_i 2^i$, $\alpha_i = 0, 1$. Knuth [9] bounds the partition function

$$P(m) = \frac{1}{4\sqrt{3m}} \exp(\pi \frac{2}{3} m) .$$

There are two possibilities for improving the bounds in $w(\Gamma_2^n, T, \Gamma_2^n)$. The first possibility is to introduce circuits to the target graph of the previous theorem, but this does not appear to give an asymptotically better bound than (4). The second possibility is to prove that balanced trees and unbalanced trees are $\leq_T$ - equivalent. This seems unlikely since combining such a result with the proof method of the previous theorem gives a polynomial sized universal tree. However, in trying to improve the bounds on $w(\Gamma_2^n, T, \Gamma_2^n)$ it may be desirable to *ignore* irregular trees, letting only very *balanced* or very *unbalanced* trees reside in the same universal data structure.

In any case, it seems unlikely that polynomial structures are possible. We are, however, far from proving this; indeed, the best known lower bound is the following

Theorem. For all $n > N$

$$w(\Gamma_2^n, T, \Gamma_2^n) > c(T)\ n \log n\ ,$$

where $c(T) > 0$ is a constant for fixed $T \geq 1$.

Certain other subcases are also of interest. Erdös, Chung, and Graham†, consider $w(S,1)$ and obtain

$$w(S,1) \leq \frac{4}{11} n^2\ .$$

The following theorem is an improvement, but is surely not the best possible bound.

Theorem

$$w(S,1) \leq \frac{2}{9} n^2$$

---

† Private Communication.

A non-trivial lower bound would clearly be desirable. Another class of interest are graphs of high genus.†† We conjecture that for graphs of fixed genus γ, it is possible to do better than the naive $\binom{n}{2}$ bound obtained by embedding in the complete graph.

Our next series of results show impressive improvements by passing to average dilations. We now get optimal constructions, even in a variety of limited settings.

We have, for instance, the

Theorem. For $\alpha > 0$,

$$a(\Gamma_2^n, \frac{1}{\alpha}, S) = O(n^{\log(2+\alpha)}) .$$

Since there is a linear lower bound on $a(\cdot,\cdot,\cdot)$, this construction is optimal. By a slight modification of the construction, this gives $a(\Gamma_2^n, A, S) = O(n)$, for all $A \geq 1$, but this result may be superceeded by the following

Theorem. For each $A \geq 1$, there is a binary tree H, such that

$$G \leq \underset{A}{\text{avg}}\ H$$

for all $G \in \Gamma_2^n$, and

$$|E(G)| = O(n) ;$$

or, in other words

$$a(\Gamma_2^n, A, \Gamma_2^n) = O(n) .$$

---

†† A graph is of genus γ if it can be embedded in a sphere with γ handles [10].

These results are related to the ability to "cut" graphs in advantageous ways. For example, a generalization of the planar separator theorem [11] to graphs of high genus, obtained by Lipton and Tarjan, gives us the following

Theorem. Let L_γ^n be the class of graphs G with genus γ and $|V(G)| = n$. Then, for all $n > N$,

$$a(L_\gamma^n, A, \Gamma_2^n) \leq c(A) \cdot n ,$$

where c(A) does not depend on n.

EXTENDED MODEL

In comparing classes of data structures (see, e.g., [6,7], the measures of "efficiency" have implicitly assumed that only sequential accessing is important. Thus, when in [6], we bound the *efficiency*, T, of an embedding of n × n array into binary trees by

$$T \geq c \log n ,$$

the function T(n) captures the dilation factor in an embedding. We now describe a generalization of this concept which recovers a certain kind of *random accessing*. Since the precise definitions are quite complex, we will settle for a less exact -- but more picturesque -- rendering. Let us assume that we have in front of us an illustration of a graph G, and also a number of friends who agree to lend us their forefingers for use in tracing the paths of the graph. Our friends oblige us as follows: We may start traversing at any vertex already visited. The traversal rule is, then, that we must either *traverse* graph edges or "jump" to a vertex pointed to

by a friend. The *time* required to traverse a sequence of vertices is then simply the number of applications of traversal rules. Notice that the result of a traversal is not necessarily a path of G. The connection between fingers and random accessing is that traversals requiring k-fingers also require k-"addresses" for the vertices pointed to.

We then say that $G \leq_{k,T} G^*$ if there is a one-one $\Phi: V(G) \to V(G^*)$, so that for every $x,y \in V(G)$ with $d_G(x,y) = m$, there is a k-finger traversal from $\Phi(x) = x^*$ to $\Phi(y) = y^*$ with time at most Δ, and $\Delta \leq Td_{G^*}(x^*, y^*)$. We have the following

Theorem. If G_n is the $n \times n$ array [7], H is a binary tree and

$$G_n \leq_{k,T(n)} H ,$$

then

$$k + T(n) \geq c \log n ,$$

where c is a constant independent of n.

OTHER TYPES OF AVERAGE EMBEDDING

The relation $\leq_A^{avg}$ may be thought of as *averaging* - with relative frequencies uniformly distributed to the edges E(G) - over the *edges* of G. We now make a more global definition which may be used to recover our intuitions about path lengths in binary trees [7]. We will essentially average our shortest paths:

$G \leq_A^{paths} G^*$ if there is an embedding $\Phi: V(G) \to V(G^*)$ such that

$$\sum_{\Phi(x),\Phi(y)} d_{G^*}(\Phi(A, \Phi(y)) \leq A \cdot \sum_{x,y} d_G(x,y) .$$

We then have the following

Theorem. For each $n \geq 0$, let A_n be the least real number such that

$$G_n \underset{A}{\leq}^{\text{paths}} H ,$$

for a binary tree H. Then

$$\lim_{n\to\infty} A_n = 0$$

Thus, we see that if the average embedding is required to work well on all shortest paths, then the embedding cost goes to zero. In a sense, then $\underset{A}{\leq}^{\text{avg}}$ "charges" more heavily than $\underset{A}{\leq}^{\text{paths}}$ for any bottlenecks.

REFERENCES

[1] P. Erdös and J. Spencer. *Probabilistic Methods in Combinatorics.* Academic Press, 1974.

[2] F. R. K. Chung and R. L. Graham. *On Graphs Which Contain Small Trees.* To appear in JCT.

[3] R. R. K. Chung, R. L. Graham, and N. Pippenger. *On Graphs Which Contain All Small Trees, II.* To appear in JCT.

[4] M. Minsky. *Semantic Information Processing.* MIT Press, 1969.

[5] L. Uhr. *Pattern Recognition Learning and Thought.* Prentice-Hall, 1973.

[6] R. J. Lipton, S. C. Eisenstat, and R. A. DeMillo. *Space and Time Hierarchies for Classes of Control and Data Structures. JACM* 23(4): 720-732, October 1976.

[7] R. A. DeMillo, S. C. Eisenstat, and R. J. Lipton. *Preserving Average Proximity In Arrays. CACM,* Vol. 21(3):228-231, March 1978.

[8] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, 1976.

[9] D. E. Knuth. *An Almost Linear Recurrence.* The Fibonacci Quarterly, Vol. 4(1):117-128, February 1966.

[10] F. Harary. *Graph Theory.* Addison-Wesley, 1972.

[11] R. Lipton and R. Tarjan. *A Planar Separator Theorem.* Stanford Research Report CS-77-627, October 1977.

SIAM J. APPL. MATH.
Vol. 36, No. 2, April 1979

0036-1399/79/3602-0001 $01.00/0

A SEPARATOR THEOREM FOR PLANAR GRAPHS*

RICHARD J. LIPTON† AND ROBERT ENDRE TARJAN‡

Abstract. Let G be any n-vertex planar graph. We prove that the vertices of G can be partitioned into three sets A, B, C such that no edge joins a vertex in A with a vertex in B, neither A nor B contains more than $2n/3$ vertices, and C contains no more than $2\sqrt{2}\sqrt{n}$ vertices. We exhibit an algorithm which finds such a partition A, B, C in $O(n)$ time.

1. Introduction. A useful method for solving many kinds of combinatorial problems is "divide-and-conquer" [1]. In this method the problem of interest is divided into two or more smaller problems. The subproblems are solved by applying the method recursively, and the subproblem solutions are combined to give the solution to the original problem. Three things are necessary for the success and efficiency of divide-and-conquer: (i) the subproblems must be of the same type as the original and independent of each other (in a suitable sense); (ii) the cost of solving the original problem given the solutions to the subproblems must be small; and (iii) the subproblems must be significantly smaller than the original. One way to guarantee that the subproblems are small is to make them all roughly the same size [1].

We wish to study general conditions under which the divide-and-conquer approach is useful. Consider problems which are defined on graphs. Let S be a class of graphs[1] closed under the subgraph relation (i.e., if $G_1 \in S$ and G_2 is a subgraph of G_1, then $G_2 \in S$). An *$f(n)$-separator theorem* for S is a theorem of the following form:

There exist constants $\alpha < 1$, $\beta > 0$ such that if G is any n-vertex graph in S, the vertices of G can be partitioned into three sets A, B, C such that no edge joins a vertex in A with a vertex in B, neither A nor B contains more than αn vertices, and C contains no more than $\beta f(n)$ vertices.

If such a theorem holds for the class of graphs S, and if the appropriate vertex partitions A, B, C can be found fast, then a number of problems defined on graphs in S can be solved efficiently using divide-and-conquer. For a given graph G in S, the sets A and B define the subproblems. The cost of combining the subproblem solutions is a function of the size of C (and thus of $f(n)$).

Previously known separator theorems include the following:

(A) Any n-vertex binary tree can be separated into two subtrees, each with no more than $2n/3$ vertices, by removing a single edge. For an application of this theorem, see [13].

(B) Any n-vertex tree can be divided into two parts, each with no more than $2n/3$ vertices, by removing a single vertex.

(C) A *grid graph* is any subgraph of the infinite two-dimensional square grid illustrated in Fig. 1. A $\sqrt{n}$-separator theorem holds for the class of grid graphs. For an application, see [5].

* Received by the editors August 10, 1977.

† Computer Science Department, Yale University, New Haven, Connecticut 06520. This research was supported in part by the U.S. Army Research Office under Grant DAAG 29-76-G-0338 and The National Science Foundation under Grant MCS 78-81486.

‡ Computer Science Department, Stanford University, Stanford, California 94305. This research was supported in part by National Science Foundation under Grant MCS-75-22870 and in part by the Office of Naval Research under Contract N00014-76-C-0688.

[1] The Appendix contains the graph-theoretic definitions used in this paper.

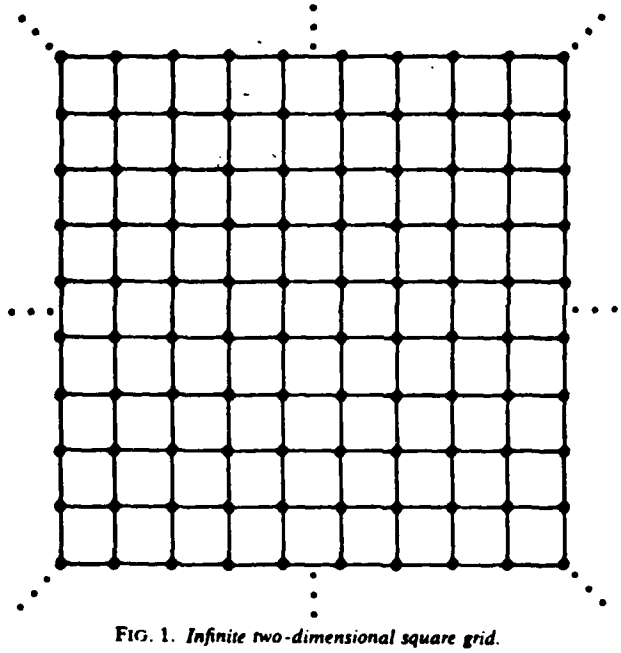


FIG. 1. *Infinite two-dimensional square grid.*

(D) A *one-tape Turing machine graph* [16] is a graph representing the computation of a one-tape Turing machine. A $\sqrt{n}$-separator theorem holds for such graphs. For an application, see [15].

One might conjecture that the class of *all* suitably sparse graphs has an $f(n)$-separator theorem for some $f(n)=o(n)$. However, the following result of Erdös Graham and Szemerédi [4] shows that this is not the case.

THEOREM C. *For every $\varepsilon > 0$ there is a positive constant $c = c(\varepsilon)$ such that almost all*[2] *graphs G with $n = (2+\varepsilon)k$ vertices and ck edges have the property that after the omission of any k vertices, a connected component of at least k vertices remains.*

Although sparsity by itself is not enough to give a useful separator theorem, planarity is. In § 2 of this paper we prove that a $\sqrt{n}$-separator theorem holds for all planar graphs. In § 3 we provide a linear-time algorithm for finding a vertex partition satisfying the theorem. This algorithm and the divide-and-conquer approach combine to give efficient algorithms for a wide range of problems on planar graphs. Section 4 mentions some of these applications, which we shall discuss more fully in a subsequent paper.

2. Separator theorems. To prove our results we need to use three facts about planarity.

THEOREM 1 (Jordan curve theorem [6]). *Let C be any closed curve in the plane. Removal of C divides the plane into exactly two connected regions, the "inside" and the "outside" of C.*

THEOREM 2 [7]. *Any n-vertex planar graph with $n \geq 3$ contains no more than $3n-6$ edges.*

[2] By "almost all" we mean that the fraction of graphs possessing the property tends with increasing n to one.

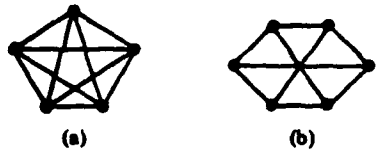


FIG. 2. *Kuratowski subgraphs*: (a) K_5; (b) $K_{3,3}$.

THEOREM 3 (Kuratowski's theorem [12]). *A graph is planar if and only if it contains neither a complete graph on five vertices* (Fig. 2(a)) *nor a complete bipartite graph on two sets of three vertices* (Fig. 2(b)) *as a generalized subgraph.*

From Kuratowski's theorem we can easily obtain the following lemma and its corollary.

LEMMA 1. *Let G be any planar graph. Shrinking any edge of G to a single vertex preserves planarity.*

Proof. Let G^* be the shrunken graph, let (x_1, x_2) be the edge shrunk, and let x be the vertex corresponding to x_1 and x_2 in G^*. If G^* is not planar then G^* contains a Kuratowski graph as a generalized subgraph. But this subgraph corresponds to a Kuratowski graph which is a generalized subgraph of G. Figure 3 illustrates the possibilities. □

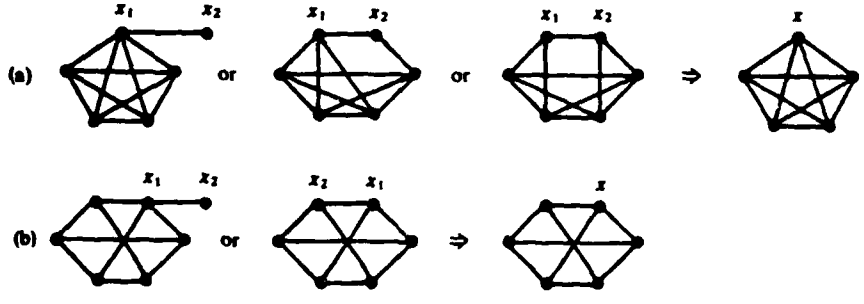


FIG. 3. *Shrinking an edge to form a Kuratowski graph. Original graph must contain a Kuratowski graph as a generalized subgraph.*

COROLLARY 1. *Let G be any planar graph. Shrinking any connected subgraph of G to a single vertex preserves planarity.*

Proof. The proof is immediate from Lemma 1 by induction on the number of vertices in the subgraph to be shrunk. □

In some applications it is useful to have a result more general than the kind of separator theorem described in the Introduction. We shall therefore consider planar graphs which have nonnegative *costs* on the vertices. We shall prove that any such graph can be separated into two parts, each with cost no more than two-thirds of the total cost, by removing $O(\sqrt{n})$ vertices. The desired separator theorem is the special case of equal-cost vertices.

LEMMA 2. *Let G be any planar graph with nonnegative vertex costs summing to no more than one. Suppose G has a spanning tree of radius r. Then the vertices of G can be partitioned into three sets A, B, C, such that no edge joins a vertex in A with a vertex in B, neither A nor B has total cost exceeding $2/3$, and C contains no more than $2r+1$ vertices, one the root of the tree.*

Proof. Assume no vertex has cost exceeding 1/3; otherwise the lemma is true. Embed G in the plane. Make each face a triangle by adding a suitable number of additional edges. Any nontree edge (including each of the added edges) forms a simple cycle with some of the tree edges. This cycle is of length at most $2r+1$ if it contains the root of the tree, at most $2r-1$ otherwise. The cycle divides the plane (and the graph) into two parts, the inside and the outside of the cycle. We claim that at least one such cycle separates the graph so that neither the inside nor the outside contains vertices whose total cost exceeds 2/3. This proves the lemma.

Proof of claim. Let (x, z) be the nontree edge whose cycle minimizes the maximum cost either inside or outside the cycle. Break ties by choosing the nontree edge whose cycle has the smallest number of faces on the same side as the maximum cost. If ties remain, choose arbitrarily.

Suppose without loss of generality that the graph is embedded so that the cost inside the (x, z) cycle is at least as great as the cost outside the cycle. If the vertices inside the cycle have total cost not exceeding 2/3, the claim is true. Suppose the vertices inside the cycle have total cost exceeding 2/3. We show by case analysis that this contradicts the choice of (x, z). Consider the face which has (x, z) as a boundary edge and lies inside the cycle. This face is a triangle; let y be its third vertex. The properties of (x, y) and (y, z) determine which of the following cases applies. Figure 4 illustrates the cases.

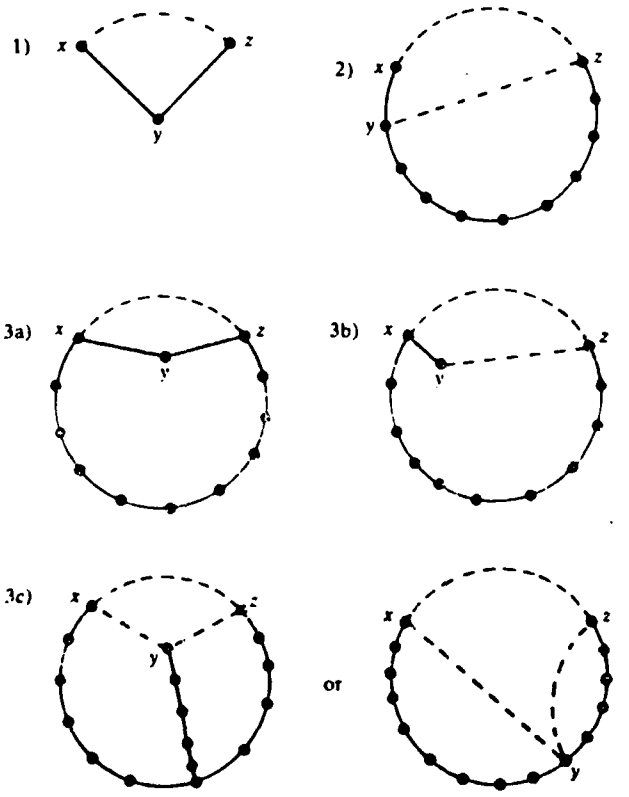


FIG. 4. *Cases for proof of Lemma 2. Solid edges are tree edges, dotted edges are nontree edges.*

1) Both (x, y) and (y, z) lie on the cycle. Then the face (x, y, z) *is* the cycle, which is impossible since vertices lie inside the cycle.

2) One of (x, y) and (y, z) (say (x, y)) lies on the cycle. Then (y, z) is a nontree edge defining a cycle which contains within it the same vertices as the original cycle but one less face. This contradicts the choice of (x, z).

3) Neither (x, y) nor (y, z) lies on the cycle.

a) Both (x, y) and (y, z) are tree edges. This is impossible since the tree itself contains no cycles.

b) One of (x, y) and (y, z) (say (x, y)) is a tree edge. Then (y, z) is a nontree edge defining a cycle which contains one less vertex (namely y) within it than the original cycle. The inside of the (y, z) cycle contains no more cost and one less face than the inside of the (x, z) cycle. Thus if the cost inside the (y, z) cycle is greater than the cost outside the cycle, (y, z) would have been chosen in place of (x, z).

On the other hand, suppose the cost inside the (y, z) cycle is no greater than the cost outside. The cost outside the (y, z) cycle is equal to the cost outside the (x, z) cycle plus the cost of y. Since both the cost outside the (x, z) cycle and the cost of y are less than $1/3$, the cost outside the (y, z) cycle is less than $2/3$, and (y, z) would have been chosen in place of (x, z).

c) Neither (x, y) nor (y, z) is a tree edge. Then each of (x, y) and (y, z) defines a cycle, and every vertex inside the (x, z) cycle is either inside the (x, y) cycle, inside the (y, z) cycle, or on the boundary of both. Of the (x, y) and (y, z) cycles, choose the one (say (x, y)) which has inside it more total cost. The (x, y) cycle has no more cost and strictly fewer faces inside it than the (x, z) cycle. Thus if the cost inside the (x, y) cycle is greater than the cost outside, (x, y) would have been chosen in place of (x, z).

On the other hand, suppose the cost inside the (x, y) cycle is no greater than the cost outside. Since the inside of the (x, z) cycle has cost exceeding $2/3$, the (x, y) cycle and its inside together have cost exceeding $1/3$, and the outside of the (x, y) cycle has cost less than $2/3$. Thus (x, y) would have been chosen in place of (x, z).

Thus all cases are impossible, and the (x, z) cycle satisfies the claim. □

LEMMA 3. *Let G be any n-vertex connected planar graph having nonnegative vertex costs summing to no more than one. Suppose that the vertices of G are partitioned into levels according to their distance from some vertex v, and that $L(l)$ denotes the number of vertices on level l. If r is the maximum distance of any vertex from v, let $r+1$ be an additional level containing no vertices. Given any two levels l_1 and l_2 such that levels 0 through l_1-1 have total cost not exceeding $2/3$ and levels l_2+1 through $r+1$ have total cost not exceeding $2/3$, it is possible to find a partition A, B, C of the vertices of G such that no edge joins a vertex in A with a vertex in B, neither A nor B has total cost exceeding $2/3$, and C contains no more than $L(l_1)+L(l_2)+\max\{0, 2(l_2-l_1-1)\}$ vertices.*

Proof. If $l_1 \geq l_2$, let A be all vertices on levels 0 through l_1-1, B all vertices on levels l_1+1 through r, and C all vertices on level l_1. Then the lemma is true. Thus suppose $l_1 < l_2$. Delete the vertices in levels l_1 and l_2 from G. This separates the remaining vertices of G into three parts (all of which may be empty): vertices on levels 0 through l_1-1, vertices on levels l_1+1 through l_2-1, and vertices on levels l_2+1 and above. The only part which can have cost exceeding $2/3$ is the middle part.

If the middle part does not have cost exceeding $2/3$, let A be the most costly part of the three, let B be the remaining two parts, and let C be the set of vertices on levels l_1 and l_2. Then the lemma is true.

Suppose the middle part has cost exceeding $2/3$. Delete all vertices on levels l_2 and above and shrink all vertices on levels l_1 and below to a single vertex of cost zero.

These operations preserve planarity by Corollary 1. The new graph has a spanning tree of radius $l_2 - l_1 - 1$ whose root corresponds to vertices on levels l_1 and below in the original graph.

Apply Lemma 2 to the new graph. Let A^*, B^*, C^* be the resulting vertex partition. Let A be the set among A^* and B^* having greater cost, let C consist of the vertices on levels l_1 and l_2 in the original graph plus the vertices in C^* minus the root of the tree, and let B contain the remaining vertices in G. By Lemma 2, A has total cost not exceeding 2/3. But $A \cup C^*$ has total cost at least 1/3, so B also has total cost not exceeding 2/3. Furthermore C contains no more than $L(l_1)+L(l_2)+2(l_2-l_1-1)$ vertices. Thus the lemma is true. □

THEOREM 4. *Let G be any n-vertex planar graph having nonnegative vertex costs summing to no more than one. Then the vertices of G can be partitioned into three sets A, B, C such that no edge joins a vertex in A with a vertex in B, neither A nor B has total cost exceeding 2/3, and C contains no more than $2\sqrt{2}\sqrt{n}$ vertices.*

Proof. Assume G is connected. Partition the vertices into levels according to their distance from some vertex v. Let $L(l)$ be the number of vertices on level l. If r is the maximum distance of any vertex from v, define additional levels -1 and $r+1$ containing no vertices.

Let l_1 be the level such that the sum of costs in levels 0 through $l_1 - 1$ is less than 1/2, but the sum of costs in levels 0 through l_1 is at least 1/2. (If no such l_1 exists, the total cost of all vertices is less than 1/2, and $B = C = \varnothing$ satisfies the theorem.) Let k be the number of vertices on levels 0 through l_1. Find a level l_0 such that $l_0 \leqq l_1$ and $|L(l_0)|+2(l_1-l_0) \leqq 2\sqrt{k}$. Find a level l_2 such that $l_1+1 \leqq l_2$ and $|L(l_2)|+2(l_2-l_1-1) \leqq 2\sqrt{n-k}$. If two such levels exist, then by Lemma 3 the vertices of G can be partitioned into three sets A, B, C such that no edge joins a vertex in A with a vertex in B, neither A nor C has cost exceeding 2/3, and C contains no more than $2(\sqrt{k}+\sqrt{n-k})$ vertices. But $2(\sqrt{k}+\sqrt{n-k}) \leqq 2(\sqrt{n/2}+\sqrt{n/2}) = 2\sqrt{2}\sqrt{n}$. Thus the theorem holds if suitable levels l_0 and l_2 exist.

Suppose a suitable level l_0 does not exist. Then, for $i \leqq l_1$, $L(i) \geqq 2\sqrt{k} - 2(l_1 - i)$. Since $L(0) = 1$, this means $1 \geqq 2\sqrt{k} - 2l_1$, and $l_1 + 1/2 \geqq \sqrt{k}$. Thus $l_1 = \lfloor l_1 + 1/2 \rfloor \geqq \lfloor \sqrt{k} \rfloor$, and

$$k = \sum_{i=0}^{l_1} L(i) \geqq \sum_{i=l_1-\lfloor\sqrt{k}\rfloor}^{l_1} 2\sqrt{k} - 2(l_1 - i) \geqq (4\sqrt{k} - 2\lfloor\sqrt{k}\rfloor)(\lfloor\sqrt{k}\rfloor + 1)/2 \geqq \sqrt{k}(\lfloor\sqrt{k}\rfloor + 1) > k.$$

This is a contradiction. A similar contradiction arises if a suitable level l_2 does not exist. This completes the proof for connected graphs.

Now suppose G is not connected. Let $G_1, G_2, \cdots, G_k$ be the connected components of G, with vertex sets $V_1, V_2, \cdots, V_k$, respectively. If no connected component has total vertex cost exceeding 1/3, let i be the minimum index such that the total cost of $V_1 \cup V_2 \cup \cdots \cup V_i$ exceeds 1/3. Let $A = V_1 \cup V_2 \cup \cdots \cup V_i$, let $B = V_{i+1} \cup V_{i+2} \cup \cdots \cup V_k$, and let $C = \varnothing$. Since i is minimum and the cost of V_i does not exceed 1/3, the cost of A does not exceed 2/3. Thus the theorem is true.

If some connected component (say G_i) has total vertex cost between 1/3 and 2/3, let $A = V_i$, $B = V_1 \cup \cdots \cup V_{i-1} \cup V_{i+1} \cup \cdots \cup V_k$, and $C = \varnothing$. Then the theorem is true.

Finally, if some connected component (say G_i) has total vertex cost exceeding 2/3, apply the above argument to G_i. Let A^*, B^*, C^* be the resulting partition. Let A be the set among A^* and B^* with greater cost, let $C = C^*$, and let B be the remaining vertices of G. Then A and B have cost not exceeding 2/3 and the theorem is true.

This proves the theorem for all planar graphs. In all cases the separator C is either empty or contained in only one connected component of G. □

COROLLARY 2 ($\sqrt{n}$-separator theorem). *Let G be any n-vertex planar graph. The vertices of G can be partitioned into three sets A, B, C such that no edge joins a vertex in A with a vertex in B, neither A nor B contains more than $2n/3$ vertices, and C contains no more than $2\sqrt{2}\sqrt{n}$ vertices.*

Proof. Assign to each vertex of G a cost of $1/n$. The corollary follows from Theorem 4. □

It is natural to ask whether the constant factor of $2/3$ in Theorem 1 can be reduced to $1/2$ if the constant factor of $2\sqrt{2}$ is allowed to increase. The answer is yes.

COROLLARY 3. *Let G be any n-vertex planar graph having nonnegative vertex costs summing to no more than one. Then the vertices of G can be partitioned into three sets A, B, C such that no edge joins a vertex in A with a vertex in B, neither A nor B has total cost exceeding $1/2$, and C contains no more than $2\sqrt{2}\sqrt{n}/(1-\sqrt{2/3})$ vertices.*

Proof. Let $G=(V, E)$ be an n-vertex planar graph. We shall define sequences of sets (A_i), (B_i), (C_i), (D_i) such that:

(i) A_i, B_i, C_i, D_i partition V,

(ii) no edge joins A_i with B_i, A_i with D_i, or B_i with D_i,

(iii) the cost of A_i is no greater than the cost of B_i and the cost of B_i is no greater than the cost of $A_i \cup C_i \cup D_i$,

(iv) $|D_i| \leq 2|D_{i-1}|/3$.

Let $A_0 = B_0 = C_0 = \varnothing$, $D_0 = V$. Then (i)–(iv) hold. If A_{i-1}, B_{i-1}, C_{i-1}, D_{i-1} have been defined and $D_{i-1} \neq \varnothing$, let G^* be the subgraph of G induced by the vertex set D_{i-1}. Let A^*, B^*, C^* be a vertex partition satisfying Corollary 2 on G^*. Without loss of generality, suppose A^* has no more cost than B^*. Let A_i be the set among $A_{i-1} \cup A^*$, B_{i-1} with less cost, let B_i be the set among $A_{i-1} \cup A^*$, B_{i-1} with greater cost, let $C_i = C_{i-1} \cup C^*$, and let $D_i = B^*$. Then (i), (ii), (iii), and (iv) hold for A_i, B_i, C_i, D_i.

Let k be the largest index for which A_k, B_k, C_k, D_k are defined. Then $D_k = \varnothing$. Let $A = A_k$, $B = B_k$, $C = C_k$. By (i), A, B, C partition V. By (ii), no edge joins a vertex in A with a vertex in B. By (iii), neither A nor B has cost exceeding $1/2$. By (iv), the total number of vertices in C is bounded by

$$\sum_{i=0}^{\infty} 2\sqrt{2}\sqrt{n}(2/3)^{i/2} = = \frac{2\sqrt{2}\sqrt{n}}{1-\sqrt{2/3}}. \quad \square$$

Another natural question is whether graphs which are "almost" planar have a $\sqrt{n}$-separator theorem. The finite element method of numerical analysis gives rise to one interesting class of almost-planar graphs. We shall extend Theorem 4 to apply to such graphs.

A *finite element graph* is any graph formed from a planar embedding of a planar graph by adding all possible diagonals to each face. (The finite element graph has a clique corresponding to each face of the embedded planar graph.) The embedded planar graph is called the *skeleton* of the finite element graph and each of its faces is an *element* of the finite element graph.

THEOREM 5. *Let G be an n-vertex finite element graph with nonnegative vertex costs summing to no more than one. Suppose no element of G has more than k boundary vertices. Then the vertices of G can be partitioned into three sets A, B, C such that no edge joins a vertex in A with a vertex in B, neither A nor B has total cost exceeding $2/3$, and C contains no more than $4\lfloor k/2 \rfloor \sqrt{n}$ vertices.*

Proof. Let G^* be the skeleton of G. Form G^{**} from G^* by inserting one new vertex into each face of G^* containing four or more vertices and connecting the new vertex to each vertex on the boundary of the face. Then G^{**} is planar. Apply Theorem 4 to G^{**}. Let A^{**}, B^{**}, C^{**} be the resulting vertex partition. This partition satisfies the theorem except that certain edges in G but not in G^{**} may join A^{**} and B^{**}. These edges are diagonals of certain faces of G^*; call these *bad faces*. Each bad face must contain one of the new vertices added to G^* to form G^{**}, and this vertex must be in C^{**}.

Form G from C^{**} by deleting all new vertices and adding to G^{**}, for each bad face, either the set of vertices in A^{**} on the boundary of the bad face, or the set of vertices in B^{**} on the boundary of the bad face, whichever is smaller. Let A be the remaining old vertices in A^{**} and let B^{**} be the remaining old vertices in B^{**}. Then no edge in G joins A and B, neither A nor B contains more than $2n/3$ vertices, and C contains no more than $2\sqrt{2}\lfloor k/2\rfloor\sqrt{n+a}$ vertices, where a is the number of faces of G^* containing four or more vertices. By use of Euler's theorem, it is not hard to show that the number of faces of G^* containing four or more vertices is at most $n-2$. Thus $|C| \leq \lfloor k/2\rfloor\sqrt{n}$, and the theorem is true. □

COROLLARY 4. *Let G be any n-vertex finite element graph. Suppose no element of G has more than k boundary vertices. The vertices of G can be partitioned into three sets A, B, C such that no edge joins a vertex in A with a vertex in B, neither A nor B contains more than $2n/3$ vertices and C contains no more than $4\lfloor k/2\rfloor\sqrt{n}$ vertices.*

The last result of this section shows that Theorem 4 and its corollaries are tight to within a constant factor; that is, if $f(n) = o(\sqrt{n})$, no $f(n)$-separator theorem holds for planar graphs.

THEOREM 6. *For any k, let $G = (V, E)$ be a $k \times k$ square grid graph* (a $k \times k$ square section of the infinite grid graph in Fig. 1). *Let A be any subset of V such that $\alpha n \leq |A| \leq n/2$, where $n = k^2$ and α is a positive constant less than $1/2$. The the number of vertices in $V-A$ adjacent to some vertex in A is at least $k \cdot \min\{1/2, \sqrt{\alpha}\}$.*

Proof. Without loss of generality, suppose that the number r of rows of G which contain vertices in A is no less than the number c of columns of G which contain vertices in A. Then $\alpha n \leq |A| \leq rc \leq r^2$ and $r \geq \sqrt{\alpha}k$.

If r^* is the number of rows of G which contain only vertices in A, then $kr^* \leq |A| \leq n/2$, and $r^* \leq k/2$. Let $S = \{x \in V: x \text{ is adjacent to a vertex of } A\}$. If $r^* = 0$, then $|S| \geq r \geq \sqrt{\alpha}k$. If $r^* \neq 0$, then $r = k$ and $|S| \geq r - r^* = k - r^* \geq k/2$. □

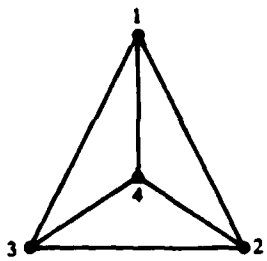
It is an open problem to determine the smallest constant factor which can replace $2\sqrt{2}$ in Theorem 4.

3. An algorithm for finding a good partition. The proof of Theorem 4 leads to an algorithm for finding a vertex partition satisfying the theorem. To make this algorithm efficient, we need a good representation of a planar embedding of a graph. For this purpose we use a list structure whose elements correspond to the edges of the graph. Stored with each edge are its endpoints and four pointers, designating the edges immediately clockwise and counter-clockwise around each of the endpoints of the edge. Stored with each vertex is some incident edge. Figure 5 gives an example of such a data structure.

PARTITIONING ALGORITHM.

Step 1. Find a planar embedding of G and construct a representation for it of the kind described above.

Time: $O(n)$, using the algorithm of [10].



Vertex incidences

1	e_1
2	e_1
3	e_2
4	e_3

Edges and neighbors

			c_1	cc_1	c_2	cc_2
e_1	1	2	e_3	e_2	e_4	e_5
e_2	1	3	e_1	e_3	e_6	e_4
e_3	1	4	e_2	e_1	e_5	e_6
e_4	2	3	e_5	e_1	e_2	e_6
e_5	2	4	e_1	e_2	e_6	e_1
e_6	3	4	e_4	e_2	e_1	e_5

FIG. 5. *Representation of an embedded planar graph.* (c = *clockwise*, cc = *counter-clockwise*.)

Step 2. Find the connected components of G and determine the cost of each one. If none has cost exceeding 2/3, construct the partition as described in the proof of Theorem 4. If some component has cost exceeding 2/3, go to Step 3.

Time: $O(n)$ [9].

Step 3. Find a breadth-first spanning tree of the most costly component. Compute the level of each vertex and the number of vertices $L(l)$ in each level l.

Time: $O(n)$.

Step 4. Find the level l_1 such that the total cost of levels 0 through $l_1 - 1$ does not exceed 1/2, but the total cost of levels 0 through l_1 does exceed 1/2. Let k be the number of vertices in levels 0 through l_1.

Time: $O(n)$.

Step 5. Find the highest level $l_0 \leqq l_1$ such that $L(l_0) + 2(l_1 - l_0) \leqq 2\sqrt{k}$. Find the lowest level $l_2 \geqq l_1 + 1$ such that $L(l_2) + 2(l_2 - l_1 - 1) \leqq 2\sqrt{n-k}$.

Time: $O(n)$.

Step 6. Delete all vertices on level l_2 and above. Construct a new vertex x to represent all vertices on levels 0 through l_0. Construct a Boolean table with one entry per vertex. Initialize to **true** the entry for each vertex on levels 0 through l_0 and

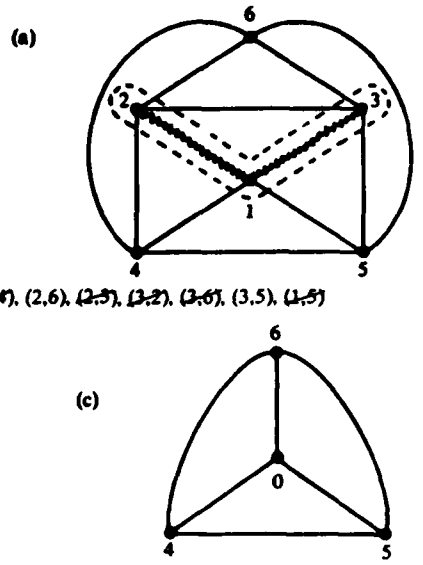


FIG. 6. *Shrinking a subtree of a planar graph.*
(a) *Original graph. Subtree denoted by* ~~~~~.
(b) *Edges scanned around subtree. Those forming loops and multiple edges in shrunken graph are crossed out.*
(c) *Shrunken graph. Vertex 0 replaces subtree.*

initialize to **false** the entry for each vertex on levels l_0+1 through l_2-1. The vertices on levels 0 through l_0 correspond to a subtree of the breadth-first spanning tree generated in Step 3. Scan the edges incident to this tree clockwise around the tree. When scanning an edge (v, w) with v in the tree, check the table entry for w. If it is **true**, delete edge (v, w). If it is **false**, change it to **true**, construct an edge (x, w), and delete edge (v, w). The result of this step is a planar representation of the shrunken graph to which Lemma 2 is to be applied. See Fig. 6.

Time: $O(n)$.

Step 7. Construct a breadth-first spanning tree rooted at x in the new graph. (This can be done be modifying the breadth-first spanning tree constructed in Step 3.) Record, for each vertex v, the parent of v in the tree, and the total cost of all descendants of v including v itself. Make all faces of the new graph into triangles by scanning the boundary of each face and adding (nontree) edges as necessary.

Time: $O(n)$.

Step 8. Choose any nontree edge (v_1, w_1). Locate the corresponding cycle by following parent pointers from v_1 and w_1. Compute the cost on each side of this cycle by scanning the tree edges incident on either side of the cycle and summing their associated costs. If (v, w) is a tree edge with v on the cycle and w not on the cycle, the cost associated with (v, w) is the descendant cost of w if v is the parent of w, and the cost of all vertices minus the descendant cost of v if w is the parent of v. Determine which side of the cycle has greater cost and call it the "inside". See Fig. 7.

Time: $O(n)$.

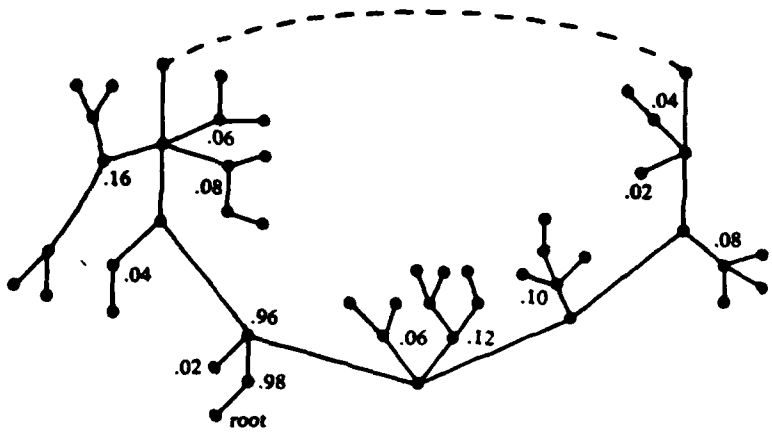


FIG. 7. *Cycle constructed in Step 8. All vertices have cost .02. Numbers on vertices are descendant costs. The total cost inside the cycle is .48, outside the cycle is .34, and on the cycle is .18.*

Step 9. Let (v_i, w_i) be the nontree edge whose cycle is the current candidate to complete the separator. If the cost inside the cycle exceeds 2/3, find a better cycle by the following method.

Locate the triangle (v_i, y, w_i) which has (v_i, w_i) as a boundary edge and lies inside the (v_i, w_i) cycle. If either (v_i, y) or (y, w_i) is a tree edge, let (v_{i+1}, w_{i+1}) be the nontree edge among (v_i, y) and (y, w_i). Compute the cost inside the (v_{i+1}, w_{i+1}) cycle from the cost inside the (v_i, w_i) cycle and the cost of v_i, y, and w_i. See Fig. 4.

If neither (v_i, y) nor (y, w_i) is a tree edge, determine the tree path from y to the (v_i, w_i) cycle by following parent pointers from y. Let z be the vertex on the (v_i, w_i) cycle reached during this search. Compute the total cost of all vertices except z on this tree path. Scan the tree edges inside the (y, w_i) cycle, alternately scanning an edge in one cycle and an edge in the other cycle. Stop scanning when all edges inside *one* of the cycles have been scanned. Compute the cost inside this cycle by summing the associated costs of all scanned edges. Use this cost, the cost inside the (v_i, w_i) cycle, and the cost on the tree path from y to z to compute the cost inside the other cycle. Let (v_{i+1}, w_{i+1}) be the edge among (v_i, y) and (y, w_i) whose cycle has more cost inside it.

Repeat Step 9 until finding a cycle whose inside has cost not exceeding 2/3.

Time: $O(n)$ (see proof below).

Step 10. Use the cycle found in Step 9 and the levels found in Step 4 to construct a satisfactory vertex partition as described in the proof of Lemma 3. Extend this partition from the connected component chosen in Step 2 to the entire graph as described in the proof of Theorem 4.

Time: $O(n)$.

This completes our presentation of the algorithm. All steps except Step 9 obviously run in $O(n)$ time. We urge readers to fill in the details of this algorithm; we content ourselves here with proving that Step 9 requires $O(n)$ time.

Proof of Step 9 time bound. Each iteration of Step 9 deletes at least one face from the inside of the current cycle. Thus Step 9 terminates after $O(n)$ iterations. The total

running time of one iteration of Step 9 is $O(1)$ plus time proportional to the length of the tree path from y to z plus time proportional to the number of edges scanned inside the (v_i, y) and (y, w_i) cycles. Each vertex on the tree path from y to z (except z) is inside the current cycle but on the boundary or outside of all subsequent cycles. For every two edges scanned during an iteration of Step 9, at least one edge is inside the current cycle but outside all subsequent cycles. It follows that the total time spent traversing tree paths and scanning edges, during all iterations of Step 9, is $O(n)$. Thus the total time spent in Step 9 is $O(n)$. □

By making minor modifications to this algorithm, one can construct an $O(n)$ time algorithm to find a vertex partition satisfying Theorem 5, and $O(n)$ time algorithms to find vertex partitions satisfying Corollary 2 and Corollary 4.

4. Applications. The separator theorem proved in § 2 allows us to obtain many new complexity results since it opens the way for efficient application of divide-and-conquer on planar graphs. We mention a few such applications here; we shall present the details in a subsequent paper.

Generalized nested dissection. Any system of linear equations whose sparsity structure corresponds to a planar or finite element graph can be solved in $O(n^{3/2})$ time and $O(n \log n)$ space. This result generalizes the nested dissection method of George [5].

Pebbling. Any n-vertex planar acyclic directed graph with maximum in-degree k can be pebbled using $O(\sqrt{n}+k \log n)$ pebbles. See [8], [16] for a description of the pebble game.

The post office problem. Knuth's "post office" problem [11] can be solved in $O((\log n)^2)$ time and $O(n)$ space. See [3], [17] for previous results.

Data structure embedding problems Any planar data structure can be efficiently embedded into a balanced binary tree. See [2], [14] for a description of the problem and some related results.

Lower bounds on Boolean circuits. Any planar circuit for computing Boolean convolution contains at least cn^2 gates for some positive constant c.

Appendix. Graph-theoretic definitions. A *graph* $G = (V, E)$ consists of a set V of *vertices* and a set E of *edges*. Each edge is an unordered pair (v, w) of distinct vertices. If (v, w) is an edge, v and w are *adjacent* and (v, w) is *incident* to both v and w. A *path* of *length* k with *endpoints* v, w is a sequence of vertices $v = v_0, v_1, v_2, \cdots, v_k = w$ such that (v_{i-1}, v_i) is an edge for $1 \leq i \leq k$. If all the vertices $v_0, v_1, \cdots, v_{k-1}$ are distinct, the path is *simple*. If $v = w$, the path is a *cycle*. The *distance* from v to w is the length of the shortest path from v to w. (The distance is infinite if v and w are not joined by a path.) The *level* of a vertex v in a graph G with respect to a fixed root r is the distance from r to v.

If $G_1 = (V_1, E_2)$ *and* $G_2 = (V_2, E_2)$ are graphs, G_1 is a *subgraph* of G_2 if $V_1 \subseteq V_2$ and $E_1 \subseteq E_2$. G_1 is a *generalized subgraph* of G_2 if $V_1 \subseteq V_2$ and there is a mapping f from E_1 into the set of paths of G_2 such that, for each edge $(v, w) \in E_1$, $f((v, w))$ has endpoints v and w, and no two paths $f((v_1, w_1))$ and $f((v_2, w_2))$ share a vertex except possibly an endpoint of both paths. If $G = (V_1, E_1)$ is a graph and $V_1 \subseteq V_2$, the graph $G_1 = (V_1, E_1)$ where $E_1 = E_2 \cap \{(v, w) | v, w \in V_1\}$ is the subgraph of G_2 *induced* by the vertex set V_1. If $G_1 = (V_1, E_1)$ is a subgraph of $G_2 = (V_2, E_2)$, then *shrinking* G_1 *to a single vertex* in G_2 means forming a new graph G_2' from G_2 by deleting from G_2 all vertices in V_1 and all their incident edges, adding a new vertex x to G_2, and adding a new edge (x, w) to G_2 for each edge $(v, w) \in E_2$ such that $v \in V_1$ and $w \notin V_1$.

A graph is *connected* if any two vertices in it are joined by a path. The *connected components* of a graph are its maximal connected subgraphs. A clique is a graph such

that any two vetices are joined by an edge. A *tree* is a connected graph containing no cycles. We shall generally assume that a tree has a distinguished vertex, called a *root*. If T is a tree with root r and v is on the (unique) simple path from r to w, v is an *ancestor* of w and w is a *descendant* of v. If in addition (v, w) is an edge of T, then v is the *parent* of w and w is a *child* of v. The *radius* of a tree is the maximum distance of any vertex from the root. A *spanning tree* T of a graph G is a subgraph of G which is a tree and which contains all the vertices of G. T is a *breadth-first spanning tree* with respect to a root r if, for any vertex v, the distance from r to v in T is equal to the distance from r to v in G.

A graph $G = (V, E)$ is *planar* if there is a one-to-one map f_1 from v into points in the plane and a map f_2 from E into simple curves in the plane such that, for each edge $(v, w) \in E$, $f_2((v, w))$ has endpoints $f_1(v)$ and $f_2(w)$, and no two curves $f_2((v_1, w_1))$, $f_2((v_2, w_2))$ share a point except possibly a common endpoint. Such a pair of maps f_1, f_2 is a *planar embedding* of G. The connected planar regions formed when the ranges of f_1 and f_2 are deleted from the plane are called the *faces* of the embedding. Each face is bounded by a curve corresponding to a cycle of G, called the *boundary* of the face. We shall sometimes not distinguish between a face and its boundary. A *diagonal* of a face is an edge (v, w) such that v and w are nonadjacent vertices on the boundary of the face.

Acknowledgments. We would like to thank Stanley Eisenstat, Rich A. DeMillo, Robert Floyd, Donald Rose, and Daniel Sleator for many helpful discussions and much thoughtful criticism.

REFERENCES

[1] A. V. AHO, J. E. HOPCROFT AND J. D. ULLMAN, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA, 1974.

[2] R. A. DEMILLO, S. C. EISENSTAT AND R. J. LIPTON, *Preserving average proximity in arrays*, Georgia Institute of Tech., Tech. Rep., Atlanta, 1976.

[3] D. DOBKIN AND R. J. LIPTON, *Multi-dimensional searching problems*, SIAM J. Comput., 5 (1976), pp. 181–186.

[4] P. ERDÖS, R. L. GRAHAM AND E. SZEMERÉDI, *On sparse graphs with dense long paths*, STAN-CS-75-504, Computer Sci. Dept., Stanford Univ., Stanford, CA, 1975.

[5] J. A. GEORGE, *Nested dissection of a regular finite element mesh*, SIAM J. Numer. Anal., 10 (1973), pp. 345–367.

[6] D. W. HALL AND G. SPENCER, *Elementary Topology*, John Wiley, New York, 1955.

[7] F. HARARY, *Graph Theory*, Addison-Wesley, Reading, MA, 1969.

[8] J. HOPCROFT, W. PAUL AND L. VALIANT, *On time versus space*, J. Assoc. Comput. Mach., 24 (1977), pp. 332–337.

[9] J. E. HOPCROFT AND R. E. TARJAN, *Efficient algorithms for graph manipulation*, Comm. ACM, 16 (1973), pp. 372–378.

[10] ——— *Efficient planarity testing*, J. Assoc. Comput. Mach., 21 (1974), pp. 549–568.

[11] D. E. KNUTH, *The Art of Computer Programming, Volume 3: Sorting and Searching*, Addison-Wesley, Reading, MA, 1973.

[12] C. KURATOWSKI, *Sur le problème des corbes gauches en topologie*, Fund. Math., 15 (1930), pp. 271–283.

[13] P. M. LEWIS, R. E. STEARNS AND J. HARTMANIS, *Memory bounds for recognition of context-free and context-sensitive languages*, IEEE Conference on Switching Theory and Logical Design, IEEE, New York, 1965, pp. 191–202.

[14] R. J. LIPTON, S. C. EISENSTAT AND R. A. DEMILLO, *Space and time hierarchies for classes of control structures and data structures*, J. Assoc. Comput. Mach., 23 (1976), pp. 710–732.

[15] M. S. PATERSON, *Tape bounds for time-bounded Turing machines*, J. Comput. System Sci., 6 (1972), pp. 116–124.

[16] W. J. PAUL, R. E. TARJAN AND J. R. CELONI, *Space bounds for a game on graphs*, Maths. Systems Theory, 10 (1977), pp. 239–251.

[17] M. J. SHAMOS, *Problems in computational geometry*, unpublished manuscript.

APPLICATIONS OF A PLANAR SEPARATOR THEOREM

Richard J. Lipton */
Computer Science Department
Yale University
New Haven, Connecticut 06520

Robert Endre Tarjan **/
Computer Science Department
Stanford University
Stanford, California 94305

August 1977

Abstract.

Any n-vertex planar graph has the property that it can be divided into components of roughly equal size by removing only $O(\sqrt{n})$ vertices. This separator theorem, in combination with a divide-and-conquer strategy, leads to many new complexity results for planar graph problems. This paper describes some of these results.

Keywords: algorithm, Boolean circuit complexity, divide-and-conquer, geometric complexity, graph embedding, lower bounds, maximum independent set, non-serial dynamic programming, pebbling, planar graphs, separator, space-time tradeoffs.

1. Introduction.

One efficient approach to solving computational problems is "divide-and-conquer" [1]. In this method, the original problem is divided into two or more smaller problems. The subproblems are solved by applying the method recursively, and the solutions to the subproblems are combined to give the solution to the original problem. Divide-and-conquer is especially efficient when the subproblems are substantially smaller than the original problem.

In [14] the following theorem is proved.

Theorem 1. Let G be any n-vertex planar graph with non-negative vertex costs summing to no more than one. Then the vertices of G can be partitioned into three sets A, B, C, such that no edge joins a vertex in A with a vertex in B, neither A nor B has total vertex cost exceeding $2/3$, and C contains no more than $2\sqrt{2}\sqrt{n}$ vertices. Furthermore A, B, C can be found in $O(n)$ time.

In the special case of equal-cost vertices, this theorem becomes

Corollary 1. Let G be any n-vertex planar graph. The vertices of G can be partitioned into three sets A, B, C, such that no edge joins a vertex in A with a vertex in B, neither A nor B contains more than $2n/3$ vertices, and C contains no more than $2\sqrt{2}\sqrt{n}$ vertices.

Theorem 1 and its corollary open the way for efficient application of divide-and-conquer to a variety of problems on planar graphs. In this paper we explore a number of such applications. Each section of the paper describes a different use of divide-and-conquer. The results range from an efficient approximation algorithm for finding maximum independent sets in planar graphs to lower bounds on the complexity of planar Boolean circuits. The last section mentions two additional applications whose description is too lengthy to be included in this paper.

2. Approximation Algorithms for NP-Complete Problems.

Divide-and-conquer in combination with Theorem 1 can be used to rapidly find good approximate solutions to certain NP-complete problems on planar graphs. As an example we consider the maximum independent set problem, which asks for a maximum number of pairwise non-adjacent vertices in a planar graph.

Theorem 2. Let G be an n-vertex planar graph with non-negative vertex costs summing to no more than one and let $0 \leq \epsilon \leq 1$. Then there is some set C of $O(\sqrt{n/\epsilon})$ vertices whose removal leaves G with no connected component of cost exceeding ϵ. Furthermore the set C can be found in $O(n \log n)$ time.

Proof. Apply the following algorithm to G.

Initialization: Let $C = \emptyset$.

General Step: Find some connected component K in G minus C with cost exceeding ϵ. Apply Corollary 1 to K, producing a partition A_1, B_1, C_1 of its vertices. Let $C = C \cup C_1$. If one of A_1 and B_1 (say A_1) has cost exceeding two-thirds the cost of K, apply Theorem 1 to the subgraph of G induced by the vertex set A_1, producing a partition A_2, B_2, C_2 of A_1. Let $C = C \cup C_2$.

Repeat the general step until G minus C has no component with cost exceeding ϵ.

The effect of one execution of the general step is to divide the component K into smaller components, each with no more than two-thirds the cost of K and each with no more than two-thirds as many vertices as K. Consider all components which arise during the course of the algorithm. Assign a level to each component as follows. If the component exists when the algorithm halts, the component has level zero. Otherwise the level of the component is one greater than the maximum level of the components formed when it is split by the general step. With this definition, any two components on the same level are vertex-disjoint.

Each level one component has cost greater than ϵ, since it is eventually split by the general step. It

*/ This research partially supported by the U. S. Army Research Office, Grant No. DAAG 29-76-G-0338.

**/ This research partially supported by National Science Foundation grant MCS-75-22870 and by the Office of Naval Research contract N00014-76-C-0688.

follows that, for $i \geq 1$, each level i component has cost at least $(3/2)^{i-1}\epsilon$ and contains at least $(3/2)^i$ vertices. Since the total cost of G is at most one, the total number of components of level i is at most $(2/3)^{i-1}/\epsilon$.

The total running time of the algorithm is $O(\Sigma\{|K| \mid K$ is a component split by the general step$\})$. Since a component of level i contains at least $(3/2)^i$ vertices, the maximum level k must satisfy $(3/2)^k \leq n$, or $k \leq \log_{3/2} n$. Since components in each level are vertex-disjoint, the total running time of the algorithm is $O(n \log_{3/2} n) = O(n \log n)$.

The total size of the set C produced by the algorithm is bounded by

$$O(\Sigma\{\sqrt{|K|} \mid K \text{ is a component split by the general step}\})$$

$$\leq O\left(\sum_{i=1}^{\lfloor \log_{3/2} n \rfloor} \max\left\{\sum_{j=1}^{\lfloor (2/3)^{i-1}/\epsilon \rfloor} \sqrt{n_j} \;\middle|\; \sum_{j=1}^{\lfloor (2/3)^{i-1}/\epsilon \rfloor} n_j \leq n \text{ and } n_j \geq 0\right\}\right)$$

$$\leq O\left(\sum_{i=1}^{\infty} \frac{(2/3)^{i-1}}{\epsilon} \sqrt{\frac{n\epsilon}{(2/3)^{i-1}}}\right)$$

$$= O\left(\sqrt{n/\epsilon} \sum_{i=0}^{\infty} (2/3)^{i/2}\right) = O(\sqrt{n/\epsilon}) \quad . \quad \square$$

The following algorithm uses Theorem 2 to find an approximately maximum independent set I in a planar graph $G = (V,E)$.

Step 1. Apply Theorem 2 to G with $\epsilon = (\log \log n)/n$ and each vertex having cost $1/n$ to find a set of vertices C containing $O(n/\sqrt{\log \log n})$ vertices whose removal leaves no connected component with more than $\log \log n$ vertices.

Step 2. In each connected component of G minus C, find a maximum independent set by checking every subset of vertices for independence. Form I as a union of maximum independent sets, one from each component.

Let I^* be a maximum independent set of G. The restriction of I^* to one of the connected components formed when C is removed from G can be no larger than the restriction of I to the same component. Thus $|I^*| - |I| = O(n/\sqrt{\log \log n})$. Since G is planar, G is four-colorable, and $|I^*| \geq n/4$. Thus $(|I^*|-|I|)/|I^*| = O(1/\sqrt{\log \log n})$, and the relative error in the size of I tends to zero with increasing n.

Step 1 of the algorithm requires $O(n \log n)$ time by Theorem 2. Step 2 requires $O(n_i 2^{n_i})$ time on a connected component of n_i vertices. The total time required by Step 2 is thus

$$O\left(\max\left\{\sum_{i=1}^{n} n_i 2^{n_i} \;\middle|\; \sum_{i=1}^{n} n_i = n \text{ and } 0 \leq n_i \leq \log \log n\right\}\right) =$$

$$O\left(\frac{n}{\log \log n} (\log \log n) 2^{\log \log n}\right) = O(n \log n) .$$

Hence the entire algorithm requires $O(n \log n)$ time.

3. Nonserial Dynamic Programming.

Many NP-complete problems, such as the maximum independent set problem, the graph coloring problem, and others, can be formulated as non-serial dynamic programming problems [2,20]. Such a problem is of the following form: minimize the objective function $f(x_1,\ldots,x_n)$, where f is given as a sum of terms $f_k(\cdot)$, each of which is a function of only a subset of the variables. We shall assume that all variables x_i take on values from the same finite set S, and that the values of the terms $f_k(\cdot)$ are given by tables. Associated with such an objective function f is an interaction graph $G = (V,E)$, containing one vertex v_i for each variable x_i in f, and an edge joining x_i and x_j for any two variables x_i and x_j which appear in a common term $f_k(\cdot)$.

By trying all possible values of the variables, a nonserial dynamic programming problem can be solved in $2^{O(n)}$ time. We shall show that if the interaction graph of the problem is planar, the problem can be solved in $2^{O(\sqrt{n})}$ time. This means that substantial savings are possible when solving typical NP-complete problems restricted to planar graphs. Note that if the interaction graph of f is planar, no term $f_k(\cdot)$ of f can contain more than four variables, since the complete graph on five vertices is not planar.

In order to describe the algorithm, we need one additional concept. The restriction of an objective function $f = \sum_{k=1}^{m} f_k$ to a set of variables $x_{i_1},\ldots,x_{i_j}$ is the objective function

$$f' = \Sigma\{f_k \mid f_k \text{ depends upon one or more of } x_{i_1},\ldots,x_{i_j}\}.$$

Given an objective function $f(x_1,\ldots,x_n) = \sum_{k=1}^{m} f_k$ and a subset S of the variables $x_1,\ldots,x_n$ which are constrained to have specific values, the following algorithm solves the problem: maximize f subject to the constraints on the variables in S. In the presentation, we do not distinguish between the variables $x_1,\ldots,x_n$ and the corresponding vertices in the interaction graph.

Step 1. If $n \leq 9$, solve the problem by exhaustively trying all possible assignments to the unconstrained variables. Otherwise, go to Step 2.

Step 2. Apply Corollary 1 to the interaction graph G of f. Let A, B, C be the resulting vertex partition. Let f_1 be the restriction of f to $A \cup C$ and let f_2 be the restriction of f to $B \cup C$. For each possible assignment of values to the variables in C-S, perform the following steps:

(a) Maximize f_1 with the given values for the variables in $C \cup S$ by applying the method recursively;

(b) Maximize f_2 with the given values for the variables in $C \cup S$ by applying the method recursively;

(c) Combine the solutions to (a) and (b) to obtain a maximum value of f with the given values for the variables in $C \cup S$.

Choose the assignment of values to variables in C-S which maximizes f and return the appropriate value of f as the solution.

The correctness of this algorithm is obvious. If $n > 9$, the algorithm solves at most $2^{O(\sqrt{n})}$ subproblems in Step 2, since C is of $O(\sqrt{n})$ size. Each subproblem contains at most $2n/3 + 2\sqrt{2}\sqrt{n} \leq 29n/30$ variables. Thus if t(n) is the running time of the algorithm, we have $t(n) \leq O(n \log n) + 2^{O(\sqrt{n})} \cdot t(29n/30)$ if $n > 9$, $t(n) = O(1)$ if $n \leq 9$. An inductive proof shows that $t(n) \leq 2^{O(\sqrt{n})}$.

4. Pebbling.

The following one-person game arises in register allocation problems [21], the conversion of recursion to iteration [16], and the study of time-space tradeoffs [4,10,18]. Let $G = (V,E)$ be a directed acyclic graph with maximum in-degree k. If (v,w) is an edge of G, v is a predecessor of w and w is a successor of v. The game involves placing pebbles on the vertices of G according to certain rules. A given step of the game consists of either placing a pebble on an empty vertex of G (called pebbling the vertex) or removing a pebble from a previously pebbled vertex. A vertex may be pebbled only if all its predecessors have pebbles. The object of the game is to successively pebble each vertex of G (in any order) subject to the constraint that at most a given number of pebbles are ever on the graph simultaneously.

It is easy to pebble any vertex of an n-vertex graph in n steps using n pebbles. We are interested in pebbling methods which use fewer than n pebbles but possibly many more than n steps. It is known that any vertex of an n-vertex graph can be pebbled with $O(n/\log n)$ pebbles [10] (where the constant depends upon the maximum in-degree), and that in general no better bound is possible [18]. We shall show that if the graph is planar, only $O(\sqrt{n})$ pebbles are necessary, generalizing a result of [18]. An example of Cook [4] shows that no better bound is possible for planar graphs.

Theorem 3. Any n-vertex planar acyclic directed graph with maximum in-degree k can be pebbled using $O(\sqrt{n} + k \log_2 n)$ pebbles.

Proof. Let $\alpha = 2\sqrt{2}$ and $\beta = 2/3$. Let G be the graph to be pebbled. Use the following recursive pebbling procedure. If $n \leq n_0$, where $n_0 = (\alpha/(1-\beta))^2$, pebble all vertices of G without deleting pebbles. If $n > n_0$, find a vertex partition A, B, C satisfying Corollary 1. Pebble the vertices of G in topological order.* To pebble a vertex v, delete all pebbles except those on C. For each predecessor u of v, let G(u) be the subgraph of G induced by the set of vertices with pebble-free paths to u. Apply the method recursively to each G(u) to pebble all predecessors of v, leaving a pebble on each such predecessor. Then pebble v.

If p(n) is the maximum number of pebbles required by this method on any n-vertex graph, then

$$p(n) = n \quad \text{if} \quad n \leq n_0,$$

$$p(n) \leq \alpha\sqrt{n} + k + p(2n/3 + \alpha\sqrt{n}) \quad \text{if} \quad n > n_0.$$

An inductive proof shows that p(n) is $O(\sqrt{n} + k \log_2 n)$. □

It is also possible to obtain a substantial reduction in pebbles while preserving a polynomial bound on the number of pebbling steps, as the following theorem shows.

Theorem 4. Any n-vertex planar acyclic directed graph with maximum in-degree k can be pebbled using $O(n^{2/3} + k)$ pebbles in $O(kn^{5/3})$ time.

Proof. Let C be a set of $O(n^{2/3})$ vertices whose removal leaves G with no weakly connected component** containing more than $n^{2/3}$ vertices. Such a set C exists by Theorem 2. The following pebbling procedure places pebbles permanently on the vertices of C. Pebble the vertices of G in topological order. To pebble a vertex v, pebble each predecessor u of v and then pebble v. To pebble a predecessor u, delete all pebbles from G except those on vertices in C or on predecessors of v. Find the weakly connected component in G minus C containing u. Pebble all vertices in this component, in topological order.

The total number of pebbles required by this strategy is $O(n^{2/3})$ to pebble vertices in C plus $n^{2/3}$ to pebble each weakly connected component plus k to pebble predecessors of the vertex v to be pebbled. The total number of pebbling steps is at most $O(k \cdot n \cdot n^{2/3}) = O(kn^{5/3})$. □

5. Lower Bounds on Boolean Circuit Size.

A Boolean circuit is an acyclic directed graph such that each vertex has in-degree zero or two, the predecessors of each vertex are ordered, and corresponding to each vertex v of in-degree two is a binary Boolean operation b_v. With each vertex of the circuit we associate a Boolean function which the vertex computes, defined as follows. With each of the k vertices v_i of in-degree zero (inputs)

---

*/ That is, in an order such that if v is a predecessor of w, v is pebbled before w.

**/ A weakly connected component of a directed graph is a connected component of the undirected graph formed by ignoring edge directions.

we associate a variable x_i and an identity function $f_{v_i}(x_i) = x_i$. With each vertex w of in-degree two having predecessors u, v we associate the function $f_w = b_w(f_u, f_v)$. The circuit computes the set of functions associated with its vertices of out-degree zero (outputs).

We are interested in obtaining lower bounds on the size (number of vertices) of Boolean circuits which compute certain common and important functions. Using Theorem 1 we can obtain such lower bounds under the assumption that the circuits are planar. Any circuit can be converted into a planar circuit by the following steps. First, embed the circuit in the plane, allowing edges to cross if necessary. Next, replace each pair of crossing edges by the crossover circuit illustrated in Figure 1. It follows that any lower bound on the size of planar circuits is also a lower bound on the total number of vertices and edge crossings in any planar representation of a non-planar circuit. In a technology for which the total number of vertices and edge crossings is a reasonable measure of cost, our lower bounds imply that it may be expensive to realize certain commonly used functions in hardware.

A superconcentrator is an acyclic directed graph with m inputs and m outputs such that any set of k inputs and any set of k outputs are joined by k vertex-disjoint paths, for all k in the range $1 \le k \le m$.

Theorem 5. Any m-input, m-output planar superconcentrator contains at least $m^2/72$ vertices.

Proof. Let G be an m-input, m-output planar superconcentrator. Assign to each input and output of G a cost of $1/(2m)$, and to every other vertex a cost of zero. Let A, B, C be a vertex partition satisfying Theorem 1 on G (ignoring edge directions). Suppose C contains p inputs and outputs. Without loss of generality, suppose that A is no more costly than B, and that A contains no more outputs than inputs. A contains between $2m/3 - p$ and $m - p/2$ inputs and outputs. Hence A contains at least $m/3 - p/2$ inputs and at most $m/2 - p/4$ outputs. B contains at least $m - p - (m/2 - p/4) = m/2 - 3p/4$ outputs. Let $k = \min\{\lceil m/3 - p/2 \rceil, \lceil m/2 - 3p/4 \rceil\}$. Since G is a superconcentrator, any set of k inputs in A and any set of k outputs in B are joined by k vertex-disjoint paths. Each such path must contain a vertex in C which is neither an input nor an output. Thus $2\sqrt{2}\sqrt{n} - p \ge \min\{m/3 - p/2, m/2 - 3p/4\} \ge m/3 - p$, and $n \ge m^2/72$. □

The property of being a superconcentrator is a little too strong to be useful in deriving lower bounds on the complexity of interesting functions. However, there are weaker properties which still require $\Omega(m^2)$ vertices. Let $G = (V,E)$ be an acyclic directed graph with m numbered inputs $v_1, v_2, \ldots, v_m$ and m numbered outputs $w_1, w_2, \ldots, w_m$. G is said to have the shifting property if, for any k in the range $1 \le k \le m$, any ℓ in the range $0 \le \ell \le m-k$, and any subset of k sources $\{v_{i_1}, \ldots, v_{i_k}\}$ such that $i_1, i_2, \ldots, i_k \le m-\ell$, there are k vertex-disjoint paths joining the set of inputs $\{v_{i_1}, \ldots, v_{i_k}\}$ with the set of outputs $\{v_{i_1+\ell}, \ldots, v_{i_k+\ell}\}$.

Theorem 6. Let G be a planar acyclic directed graph with the shifting property. Then G contains at least $\lfloor m/2 \rfloor^2/162$ vertices.

Proof. Suppose that G contains n vertices. Assign a cost of $1/m$ to each of the first $\lfloor m/2 \rfloor$ inputs and to each of the last $\lfloor m/2 \rfloor$ outputs of G, and a cost of zero to every other vertex of G. Call the first $\lfloor m/2 \rfloor$ inputs and the last $\lfloor m/2 \rfloor$ outputs of G costly. Let A, B, C be a vertex partition satisfying Theorem 1 on G (ignoring edge directions).

Without loss of generality, suppose that A is no more costly than B, and that A contains no more costly outputs than costly inputs. Let A' be the set of costly inputs in A, B' the set of costly outputs in B, p the number of costly inputs and outputs in C, and q the number of costly inputs and outputs in A. Then $2\lfloor m/2 \rfloor/3 - p \le q \le \lfloor m/2 \rfloor - p/2$. Hence $|A'| \ge q/2 \ge \lfloor m/2 \rfloor/3 - p/2$. Also

$$\begin{aligned} |A'|\cdot|B'| &\ge |A'|\cdot(\lfloor m/2 \rfloor - p - (q - |A'|)) \\ &\ge q/2 \cdot (\lfloor m/2 \rfloor - p - q/2) \\ &\ge (\lfloor m/2 \rfloor/3 - p/2)(\lfloor m/2 \rfloor - p - \\ &\qquad \lfloor m/2 \rfloor/3 + p/2) \\ &= (\lfloor m/2 \rfloor/3 - p/2)(2\lfloor m/2 \rfloor/3 - p/2) \\ &\ge 2\lfloor m/2 \rfloor^2/9 - p\lfloor m/2 \rfloor/2 . \end{aligned}$$

For $v_i \in A'$, $w_j \in B'$, and ℓ in the range $1 \le \ell \le \lfloor m/2 \rfloor$, call v_i, w_j, ℓ a match if $j - i = \ell$. For every $v_i \in A'$ and $w_j \in B'$ there is exactly one value of ℓ which produces a match; hence the total number of matches for all possible v_i, w_j, ℓ is $|A'|\cdot|B'| \ge 2\lfloor m/2 \rfloor^2/9 - p\lfloor m/2 \rfloor/2$. Since there are only $\lfloor m/2 \rfloor$ values of ℓ, some value of ℓ produces at least $2\lfloor m/2 \rfloor/9 - p/2$ matches. Thus, for $k = 2\lfloor m/2 \rfloor/9 - p/2$, there is some value of ℓ add some set of k inputs $A'' = \{v_{i_1}, v_{i_2}, \ldots, v_{i_k}\} \subseteq A'$ such that $B'' = \{w_{i_1+\ell}, w_{i_2+\ell}, \ldots, w_{i_k+\ell}\} \subseteq B'$.

Since G has the shifting property, there must be k vertex-disjoint paths between A'' and B''. But each such path must contain a vertex of C which is neither an input nor an output. Hence $2\sqrt{2}\sqrt{n} - p \ge 2\lfloor m/2 \rfloor/9 - p/2$, and $n \ge \lfloor m/2 \rfloor^2/162$. □

A shifting circuit is a Boolean circuit with m primary inputs $x_1, x_2, \ldots, x_m$, zero or more auxiliary inputs, and m outputs $z_1, z_2, \ldots, z_m$, such that, for any k in the range $0 \le k \le m$, there is some assignment of the constants 0, 1 to the auxiliary inputs so that output z_{i+1} computes the identity function x_i, for $0 \le i \le m-k$. The Boolean convolution of two Boolean vectors $(x_1, x_2, \ldots, x_m)$ and $(y_1, y_2, \ldots, y_m)$ is the vector $(z_2, z_3, \ldots, z_{2m})$ given by $z_k = \sum_{i+j=k} x_i y_j$.

Corollary 2. Any planar shifting circuit has at least $\lfloor m/2 \rfloor^2/162$ vertices.

Proof. Any shifting circuit has the shifting property. See [23,24]. □

Corollary 3. Any planar circuit for computing Boolean convolution has at least $\lfloor m/2 \rfloor^2/162$ vertices.

Proof. A circuit for computing Boolean convolution is a shifting circuit if we regard $x_1,\ldots,x_m$ as the primary inputs and $z_2,\ldots,z_{m+1}$ as the outputs. □

Corollary 4. Any planar circuit for computing the product of two m bit binary integers has at least $\lfloor m/2 \rfloor^2/162$ vertices.

Proof. A circuit for multiplying two m-bit binary integers is a shifting circuit. □

The last result of this section is an $\Omega(m^4)$ lower bound on the size of any planar circuit for multiplying two $m \times m$ Boolean matrices. We shall assume that the inputs are x_{ij}, y_{ij} for $1 \le i,j \le m$ and the outputs are z_{ij} for $1 \le i,j \le m$. The circuit computes $Z = X \cdot Y$, where $Z = (z_{ij})$, $X = (x_{ij})$, and $Y = (y_{ij})$. We use the following property of circuits for multiplying Boolean matrices, called the matrix concentration property [23,24]. For any k in the range $1 \le k \le n^2$, any set $\{x_{i_r j_r} \mid 1 \le r \le k\}$ of k inputs from X, and any permutation σ of the integers one through n, there exist k vertex-disjoint paths from $\{x_{i_r j_r} \mid 1 \le r \le k\}$ to $\{z_{i_r \sigma(j_r)} \mid 1 \le r \le k\}$. Similarly, for any k in the range $1 \le k \le n^2$, any set $\{y_{i_r j_r} \mid 1 \le r \le k\}$ of k inputs from Y, and any permutation σ of one through n, there exist k vertex-disjoint paths from $\{y_{i_r j_r} \mid 1 \le r \le k\}$ to $\{z_{\sigma(i_r) j_r} \mid 1 \le r \le k\}$.

Theorem 7. Any planar circuit G for multiplying two $m \times m$ Boolean matrices contains at least cm^4 vertices, for some positive constant c.

Proof. This proof is somewhat involved, and we make no attempt to maximize the constant factor. Suppose G contains n vertices, and that m is even. Assign a cost of $1/(4m^2)$ to each input x_{ij} and each input y_{ij}, a cost of $1/(2m^2)$ to each output z_{ij}, and a cost of zero to every other vertex. There is a partition A, B, C of the vertices of G such that neither A nor B has total cost exceeding $1/2$, no edge joins a vertex in A with a vertex in B, and C contains no more than $2\sqrt{2}\sqrt{n}/(1 - \sqrt{2/3}) = c_1\sqrt{n}$ vertices. This is a corollary of Theorem 1; see [14]. Without loss of generality, suppose that B contains no fewer outputs than A, and that A contains no fewer inputs x_{ij} than inputs y_{ij}. Then B contains at least $(m^2 - c_1\sqrt{n})/2$ outputs, which contribute at least $1/4 - c_1\sqrt{n}/(4m^2)$ to the cost of B. Thus inputs contribute at most $1/4 - c_1\sqrt{n}/(4m^2)$ to the cost of B, and B contains at most $m^2 + c_1\sqrt{n}$ inputs. A contains at least $2m^2 - (m^2 + c_1\sqrt{n}) - c_1\sqrt{n} = m^2 - 2c_1\sqrt{n}$ inputs, of which at least $m^2/2 - c_1\sqrt{n}$ are inputs x_{ij}. One of the following cases must hold.

Case 1. A contains at least $3m^2/5$ inputs x_{ij}. Let p be the number of columns of X which contain at least $4m/7$ elements of A. Then $pm + (m-p)(4m/7) \ge 3m^2/5$, and $p \ge m/15$. Let q be the number of columns of Z which contain at least $4m/9$ elements of B. Then $qm + (m-q)(4m/9) \ge m^2/2 - c_1\sqrt{n}/2$, and $q \ge m/10 - 9c_1\sqrt{n}/(10m)$.

Let $k = \min\{m/15, m/10 - 9c_1\sqrt{n}/(10m)\}$. Choose any k columns of X, each of which contains at least $4m/7$ elements of A. Match each such column of X with a column of Z which contains at least $4m/9$ elements of B. For each pair of matched columns x_{*i}, z_{*j}, select a set of $4m/7 + 4m/9 - m = m/63$ rows ℓ such that $x_{\ell i}$ is in A and $z_{\ell j}$ is in B. Such a selection gives a set of $km/63$ elements in $X \cap A$ and a set of $km/63$ elements in $Z \cap B$ which must be joined by $km/63$ vertex-disjoint paths, since G has the matrix concentration property. Each such path must contain a vertex of C. Thus $km/63 \le c_1\sqrt{n}$, which means either $m^2/(15 \cdot 63c_1) \le c_1\sqrt{n}$ (i.e., $(m^2/(15 \cdot 63c_1))^2 \le n$) or $m/63(m/10 - 9c_1\sqrt{n}/(10m)) \le c_1\sqrt{n}$ (i.e., $(m^2/(9 \cdot 69c_1))^2 \le n$).

Case 2. A contains fewer than $3m^2/5$ inputs x_{ij}. Then A contains at least $2m^2/5 - 2c_1\sqrt{n}$ inputs y_{ij}. Let S be the set of $m/2$ columns of Z which contain the most elements in B.

Subcase 2a. S contains at least $3m^2/10$ elements in B. Let p be the number of columns of X which contain at least $4m/9$ elements of A. Then $pm + 4(m-p)m/9 \ge m^2/2 - c_1\sqrt{n}$, and $p \ge m/10 - 9c_1\sqrt{n}/(5m)$. Let q be the number of columns of Z which contain at least $4m/7$ elements of B. Then $qm + 4(m/2 - q)m/7 \ge 3m^2/10$, and $q \ge m/30$. A proof similar to that in Case 1 shows that $n \ge cm^4$ for some positive constant c.

Subcase 2b. S contains fewer than $3m^2/10$ elements in B. Then the $m/2$ columns of Z not in S contain at least $m^2/5 - c_1\sqrt{n}/2$ elements in B. Let q be the number of columns of Z not in S which contain at least $m/10$ elements in B. Then $qm + (m/2 - q)(m/10) \ge m^2/5 - c_1\sqrt{n}/2$, and $q \ge m/6 - 5c_1\sqrt{n}/(9m)$. If $0 \ge q \ge m/6 - 5c_1\sqrt{n}/(9m)$, then $(3m^2/(10c_1))^2 \ge n$. Hence assume $q > 0$. Then all columns in S must contain at least $m/10$ elements in B, and $2m/3 - 5c_1\sqrt{n}/(9m)$ columns of

Z must contain at least m/10 elements in B .

Let p be the number of columns of Y which contain at least m/25 elements of A . Then $pm + (m-p)(m/25) \geq 2m^2/5 - 2c_1\sqrt{n}$, and $p \geq 3m/8 - 25c_1\sqrt{n}/(12m)$.

For any input $y_{ij} \in A$ and integer ℓ in the range $-n+1 \leq \ell \leq n-1$, call y_{ij} , ℓ a match if $z_{i+\ell,j} \in B$. By the previous computations, there are at least $2m/3 - 5c_1\sqrt{n}/(9m) + 3m/8 - 25c_1\sqrt{n}/(12m) - m = m/25 - 95c_1\sqrt{n}/(36m) = m/25 - c_1\sqrt{n}/m$ columns j such that y_{*j} contains m/25 elements of A and z_{*j} contains m/10 elements of B . Each such column produces $m^2/250$ matches; thus the total number of matches is at least $m^3/6250 - mc_1\sqrt{n}/250$. Since there are only 2m-1 values of ℓ , some value of ℓ produces at least $k = m^2/12,500 - c_2\sqrt{n}/500$ matches. Since G has the matrix concentration property, this set of matches corresponds to a set of k elements in $Y \cap A$ and a set of k elements in $Z \cap B$ which must be joined by k vertex-disjoint paths. Each such path must contain a vertex in C . Thus $k \leq c_1\sqrt{n}$, which means $m^4/(12,500(c_1 + c_2/500))^2 \leq n$.

In all cases $n \geq cm^4$ for some positive constant c . Choosing the minimum c over all cases gives the theorem for even m . The theorem for odd m follows immediately. □

The bounds in Theorems 5 - 7 and Corollaries 2 - 4 are tight to within a constant factor. We leave the proof of this fact as an exercise.

6. Embedding of Data Structures.

Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be undirected graphs. An embedding of G_1 in G_2 is a one-to-one map $\phi: V_1 \to V_2$. The worst-case proximity of the embedding is $\max\{d_2(\phi(v),\phi(w)) \mid \{v,w\} \in E_1\}$, where $d_2(x,y)$ denotes the distance between x and y in G_2 . The average proximity of the embedding is $\frac{1}{|E_1|} \sum \{d_2(\phi(v),\phi(w)) \mid \{v,w\} \in E_1\}$. These notions arise in the following context. Suppose we wish to represent some kind of data structure by another kind of data structure, in such a way that if two records are logically adjacent in the first data structure, their representations are close together in the second. We can model the data structures by undirected graphs, with vertices denoting records and edges denoting logical adjacencies. The representation problem is then a graph embedding problem in which we wish to minimize worst-case or average proximity. See [5,13,19] for research in this area.

Theorem 8. Any planar graph with maximum degree k can be embedded in a binary tree so that the average proximity is a constant depending only upon k .

Proof. Let G be an n-vertex planar graph. Embed G in a binary tree T by using the following recursive procedure. If G has one vertex v , let T be the tree of one vertex, the image of v . Otherwise, apply Corollary 1 to find a partition A, B, C of the vertices of G . Let v be any vertex in C (if C is empty, let v be any vertex). Embed the subgraph of G induced by $A \cup C - \{v\}$ in a binary tree T_1 by applying the method recursively. Embed the subgraph of G induced by B in a binary tree T_2 by applying the method recursively. Let T consist of a root (the image of v) with two children, the root of T_1 and the root of T_2 . Note that the tree T constructed in this way has exactly n vertices.

Let h(n) be the maximum depth of a tree T of n vertices produced by this algorithm. Then

$$h(n) \leq 9 \quad \text{if } n \leq 9 ,$$

$$h(n) \leq h(2n/3 + 2\sqrt{2}\sqrt{n} - 1) \leq h(29n/30) \quad \text{if } n > 9 .$$

It follows that h(n) is O(log n) .

Let $G = (V,E)$ be an n-vertex graph to which the algorithm is applied, let G_1 be the subgraph of G induced by $A \cup C$, and let G_2 be the subgraph induced by B . If $s(G) = \sum \{d_2(\phi(v),\phi(w)) \mid (v,w) \in E\}$, then $s(G) = 0$ if $n = 1$, and $s(G) \leq s(G_1) + s(G_2) + k|C|\,h(n)$ if $n > 1$. This follows from the fact that any edge of G not in G_1 or G_2 must be incident to a vertex of C .

If s(n) is the maximum value of s(G) for any n-vertex graph G , then

$$s(1) = 0 ;$$

$$s(n) \leq \max\{s(i) + s(n-i) + ck\sqrt{n}\log n \mid n/3 - 2\sqrt{2}\sqrt{n} \leq i \leq 2n/3 + 2\sqrt{2}\sqrt{n}\}$$

if $n > 1$, for some positive constant c .

An inductive proof shows that s(n) is O(kn) .

If G is a connected n-vertex graph embedded by the algorithm, then G contains at least n-1 edges, and the average proximity is O(k) . If G is not connected, embedding each connected component separately and combining the resulting trees arbitrarily achieves an O(k) average proximity. □

It is natural to ask whether any graph of bounded degree can be embedded in a binary tree with O(1) average proximity. (Graphs of unbounded degree cannot be so embedded; the star of Figure 2 requires $\Omega(\log n)$ proximity.) Such is not the case, and in fact the property of being embeddable in a binary tree with O(1) average proximity is closely related to the property of having a good separator.

To make this statement more precise, let S be a class of graphs. The class S has an f(n) -separator theorem if there exist constants $\alpha < 1$, $\beta > 0$ such that the vertices of any n-vertex graph in S can be partitioned into three sets A, B, C such that $|A|, |B| \leq \alpha n$, $|C| \leq \beta f(n)$, and no vertex in A is adjacent to any vertex in B .

Let S be any class of graphs of bounded degree closed under the subgraph relation (i.e., if $G_2 \in S$ and G_1 is a subgraph of G_2 , then $G_1 \in S$). Suppose S satisfies an $ng(n)/(\log n)^2$ separator theorem for

some non-decreasing function $g(n)$. Using the idea in the proof of Theorem 8, it is not hard to show that any graph in S can be embedded in a binary tree with $O(g(n))$ average proximity. Conversely, suppose any graph in a class S can be embedded in a binary tree with $O(g(n))$ average proximity. Then S satisfies an $ng(n)/\log n$ separator theorem. In particular, if S satisfies no $o(n)$-separator theorem, then embedding the graphs of S in binary trees requires $\Omega(\log n)$ average proximity. Erdös, Graham, and Szemerédi [7] have shown the existence of a class of graphs of bounded degree having no $o(n)$-separator theorem.

7. The Post Office Problem.

In [11], Knuth mentions the following problem: given n points (post offices) in the plane; determine, for any new point (house), which post office it is nearest. Any preprocessing of the post offices is allowed before the houses are processed. Shamos [22] gives an $O(\log n)$-time, $O(n^2)$-space algorithm and an $O((\log n)^2)$-time, $O(n \log n)$-space algorithm. See also [6]. Using Theorem 2 we can give a solution which requires $O(\log n)$ time and $O(n)$ space, both minimum if only binary decisions are allowed.

A polygon is a connected, open planar region bounded by a finite set of line segments. (For convenience, we allow the point at infinity to be an endpoint of a line segment; thus a line is a line segment.) A polygon partition of the plane is a partition of the plane into polygons and bounding line segments. A triangulation of the plane is a polygon partition, all of whose polygons are bounded by three line segments. A triangulation of a polygon partition is a refinement of the partition into a triangulation. Two polygons in a polygon partition are adjacent if their boundaries share a line segment. A set of polygons is connected if any two polygons in the set are joined by a sequence of adjacent polygons.

We shall solve the following triangle problem: given an n-triangle triangulation and a point, determine which triangle or line segment of the triangulation contains the point. The post office problem can be reformulated as triangle problem; the set of points closest to each post office forms a polygon [22]. We shall make use of the following lemma, which we do not prove.

Lemma 1. Any n-polygon partition has a refinement whose total number of triangles is bounded by n plus the number of line segments bounding non-triangles plus a constant (a line segment bounding two non-triangles counts twice in this bound).

We shall build up a sequence of more and more complicated (but more and more efficient) algorithms, the last of which is the desired one.

Theorem 9. Given an $O(\log n)$-time, $O(n^{1+\epsilon})$-space algorithm for the triangle problem with $\epsilon > 0$, one can construct an $O(\log n)$-time, $O(n^{1+2\epsilon/3})$-space algorithm.

Proof. Let T be a triangulation and v be a vertex for which the triangle problem is to be solved. By Theorem 2 there is a set of $O(n^{2/3})$ triangles C_0 whose removal from T leaves no connected set of more than $O(n^{2/3})$ triangles.

Merge pairs of adjacent triangles which are not in C_0 to form a polygon partition P_0. P_0 contains at most $O(n^{2/3})$ line segments, since each such line segment must be a bounding segment of a triangle in T. Find a triangulation T_0 of P_0 with $O(n^{2/3})$ triangles, which exists by Lemma 1. Using the given algorithm, determine which triangle or line segment of T_0 contains v.

If v is in some triangle of C_0, the problem is solved. Otherwise, v is known to be in some connected set C_i of triangles in T minus C_0. Merge pairs of adjacent triangles which are not in C_i to form a polygon partition P_i. Since P_i contains at most $O(n^{2/3})$ line segments, there is a triangulation T_i of P_i with $O(n^{2/3})$ triangles. Using the given algorithm, determine which triangle or line segment of T_i contains v. This solves the problem.

The sets C_i, polygon partitions P_i, and triangulations T_i are all precomputed. Thus the time required by the algorithm is $O(\log n^{2/3})$ to discover which triangle of T_0 contains v, plus $O(\log n^{2/3})$ to discover which triangle of T_i contains v. The total time is thus $O(\log n)$. The total space is $\sum_i O(|T_i|^\alpha) \le O(n^{1+2\epsilon/3})$. □

Corollary 5. For any $\epsilon > 0$ there is an $O(\log n)$-time, $O(n^{1+\epsilon})$-space algorithm for the triangle problem.

Proof. Immediate from Theorem 9, using the $O(\log n)$-time, $O(n^2)$-space algorithm of [22] as a starting point. □

Theorem 10. There is an $O(\log n)$-time, $O(n)$-space solution to the triangle problem.

Proof. Let T be a triangulation and v a vertex for which the triangle problem is to be solved. If T contains no more than n_0 triangles, where n_0 is a sufficiently large constant, determine which triangle contains v by testing v against each line segment bounding a triangle of T. Otherwise, let C be a set of $O(n^{3/5})$ triangles whose removal from T leaves no connected set of more than $O(n^{4/5})$ triangles. Group the connected sets of triangles in T minus C_0 into sets C_i, each containing within a constant factor of $n^{4/5}$ triangles.

Merge pairs of adjacent triangles which are not in C_0 to form a polygon partition P_0. P_0 contains at most $O(n^{3/5})$ line segments. Find a triangulation T_0 of P_0 with $O(n^{3/5})$ triangles. Using an $O(\log n)$-time, $O(n^{7/6})$-space algorithm, determine which triangle of T_0 contains v.

If v is some triangle of C_0, the problem is solved. Otherwise v is known to be in some set C_i. Merge pairs of adjacent triangles which are not in C_i to form a polygon partition P_i. Each line segment bounding a non-triangular polygon of P_i must bound a triangle of C_0. Thus there is a triangulation T_i

of F_i containing $|C_i| + O(n^{3/5})$ triangles. Apply the algorithm recursively to discover which triangle of T_i contains v. This solves the problem.

The sets C_i, polygon partitions P_i, and triangulations T_i are all precomputed. If $t(n)$ is the worst-case time required by the algorithm on an n-triangle triangulation, then

$$t(n) = O(1) \quad \text{if } n \le n_0 ,$$

$$t(n) = t(O(n^{4/5})) + O(\log n) \quad \text{otherwise.}$$

An inductive proof shows that $t(n)$ is $O(\log n)$ if n_0 is chosen sufficiently large.

If $s(n)$ is the worst-case storage space required by the algorithm on an n-triangle triangulation, then

$$s(t) = O(1) \quad \text{if } n \le n_0 ,$$

$$s(n) \le O(n^{7/10}) + \max\{\textstyle\sum s(n_i + O(n^{3/5})) \mid \sum n_i \le n \text{ and } c_1 n^{4/5} \le n_i \le c_2 n^{4/5}\}$$

for some positive constants c_1 and c_2.

An inductive proof shows that $s(n)$ is $O(n)$. □

The preprocessing time required by the algorithm of Theorem 10 is $O(n \log n)$. See [22]. We do not advocate this algorithm as a practical one, but its existence suggests that there may be a practical algorithm with an $O(\log n)$ time bound and $O(n)$ space bound.

8. Other Applications.

As illustrated in this paper, Theorem 1 and its corollaries have many interesting applications, and the paper does not exhaust them. We have obtained two additional results which require fuller discussion than is possible here. One is the application of Theorem 1 to Gaussian elimination. George [8] has proposed an $O(n \log n)$ -space, $O(n^{3/2})$ -time method of carrying out Gaussian elimination on a system of equations whose sparsity structure corresponds to a $\sqrt{n} \times \sqrt{n}$ square grid. We can generalize his method so that it applies to any system of equations whose sparsity structure corresponds to a planar or almost-planar graph. Such systems arise in the solution of two-dimensional finite-element problems [15]. We shall discuss this application in a subsequent paper; we hope that it will prove of practical, as well as theoretical, value.

Another application involves the power of non-determinism in one-tape Turing machines. We can prove that any non-deterministic $t(n)$ -time-bounded one-tape Turing machine can be simulated by a $t(n)^{\gamma}$ alternating one-tape Turing machine with a constant number of alternations, where $\gamma < 1$ is a suitable constant and $t(n)$ satisfies certain reasonable restrictions. Alternation generalizes the concept of non-determinism and is discussed in [3,12]. Our result strengthens Paterson's space-efficient simulation of one-tape Turing machines [17].

References.

[1] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, The Design and Analysis of Efficient Computer Algorithms, Addison-Wesley, Reading, Mass., 1974.

[2] U. Bertele and F. Brioschi, Nonserial Dynamic Programming, Academic Press, New York, 1972.

[3] A. K. Chandra and L. J. Stockmeyer, "Alternation," Proc. Seventeenth Annual Symp. on Foundations of Computer Science (1976), 98-108.

[4] S. A. Cook, "An observation on time-storage trade-off," Proc. Fifth Annual ACM Symp. on Theory of Computing (1973), 29-33.

[5] R. A. DeMillo, S. C. Eisenstat, and R. J. Lipton, "Preserving average proximity in arrays," School of Information and Computer Science, Georgia Institute of Technology (1976).

[6] D. Dobkin and R. J. Lipton, "Multidimensional searching problems," SIAM J. Comput. 5 (1976), 181-186.

[7] P. Erdös, R. L. Graham, and E. Szemerédi, "On sparse graphs with dense long paths," STAN-CS-75-504, Computer Science Dep't Stanford University (1975).

[8] J. A. George, "Nested dissection of a regular finite element mesh," SIAM J. Numer. Anal. 10 (1973), 345-363.

[9] L. Goldschlager, "The monotone and planar circuit value problems are log space complete for P," ACM SIGACT News 9, 2 (1977), 25-29.

[10] J. Hopcroft, W. Paul, and L. Valiant, "On time versus space," Journal ACM 24 (1977), 332-337.

[11] D. E. Knuth, The Art of Computer Programming, Volume 3: Sorting and Searching, Addison-Wesley, Reading, Mass., 1973.

[12] D. Kozen, "On parallelism in Turing machines," Proc. Seventeenth Annual Symp. on Foundations of Computer Science (1976), 89-97.

[13] R. J. Lipton, S. C. Eisenstat, and R. A. DeMillo, "Space and time hierarchies for control structures and data structures," Journal ACM 23 (1976), 720-732.

[14] R. J. Lipton and R. E. Tarjan, "A separator theorem for planar graphs," to appear.

[15] H. C. Martin and G. F. Carey, Introduction to Finite Element Analysis, McGraw-Hill, New York, 1973.

[16] M. S. Paterson and C. E. Hewitt, "Comparative schematology," Record of Project MAC Conf. on Concurrent Systems and Parallel Computation (1970), 119-128.

[17] M. S. Paterson, "Tape bounds for time-bounded Turing machines," Journal Computer and System Sciences 6 (1972), 116-124.

[18] W. J. Paul, R. E. Tarjan, and J. R. Celoni, "Space bounds for a game on graphs," Math. Systems Theory 10 (1977), 239-251.

[19] A. L. Rosenberg, "Managing storage for extendible arrays," SIAM J. Comput. 4 (1975), 287-306.

[20] A. Rosenthal, "Nonserial dynamic programming is optimal," Proc. Ninth Annual ACM Symp. on Theory of Computing (1977), 98-105.

[21] R. Sethi, "Complete register allocation problems," SIAM J. Comput. 4 (1975), 226-248.

[22] M. J. Shamos, "Geometric complexity," Proc. Seventh Annual ACM Symp. on Theory of Computing (1975), 224-233.

[23] L. G. Valiant, "On non-linear lower bounds in computational complexity," Proc. Seventh Annual ACM Symp. on Theory of Computing (1975), 45-53.

[24] L. G. Valiant, "Graph-theoretic arguments in low-level complexity," Computer Science Dept., University of Edinburgh (1977).

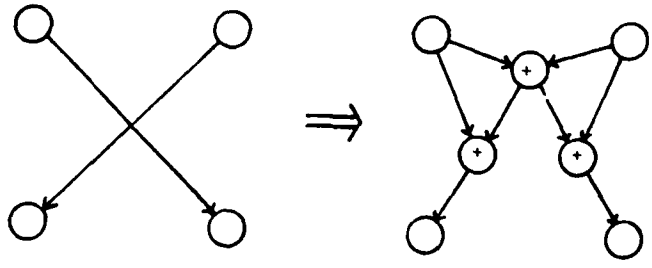


Figure 1. Elimination of a crossover by use of three "exclusive or" gates. Reference [9] contains a crossover circuit which uses only "and" and "not".

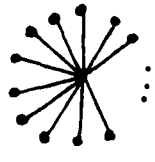


Figure 2. A star.

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER (14) GIT-ICS-79/12 — 2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) (6) Combinatorial Graph Embedding	5. TYPE OF REPORT & PERIOD COVERED (9) Final rept.s 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) (10) R.A. DeMillo, S.C. Eisenstat, R.J. Lipton	8. CONTRACT OR GRANT NUMBER(s) (15) DAAG29-76-G-0338
9. PERFORMING ORGANIZATION NAME AND ADDRESS Georgia Institute of Technology School of Information and Computer Science Atlanta, Georgia 30332	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS (12) 85
11. CONTROLLING OFFICE NAME AND ADDRESS U. S. Army Research Office P. O. Box 12211 Research Triangle Park, NC 27709	12. REPORT DATE (11) Jan 80 13. NUMBER OF PAGES 82
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) (18) ARO	15. SECURITY CLASS. (of this report) Unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

(19) 14690.10-EL

18. SUPPLEMENTARY NOTES

The view, opinions, and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy, or decision, unless so designated by other documentation.

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

an element of a subset of

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Let G, G' be directed graphs. A combinatorial embedding of G into G' is an identification of each $x \in V(G)$ with a set of vertices $S \subseteq V(G')$ such that each S is bounded in size by a constant independent of $\langle V(G) \rangle$ and each arc in G is carried into a directed path of length bounded by a constant independent of $\langle V(G) \rangle$. This concept ~~(first defined in [A])~~ has formed the basis for a number of theoretical studies ~~supported by ARO Contract No.~~ DAAG29-76-G-0338~~,~~ and the papers collected here are representative of - with one major exception - the state-of-the-art with regard to graph embedding.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

Unclassified 410044

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)